MASTER EN SCIENCES INFORMATIQUES

Etude du multicast sur Internet et réalisation d'un simulateur PIM SM (Protocol independent multicast sparse mode)

Dupuis, Eric

Award date:
2000

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

Etude du Multicast sur Internet

Et

Réalisation d'un simulateur PIM SM
(Protocol Independent Multicast Sparse Mode)

Dupuis Eric

Mémoire présenté en vue de l'obtention du grade de

Licencié en Informatique

Année Académique 1999 – 2000

Promoteur : Olivier Bonaventure

Résumé

Ce document n'est pas un traité dans lequel vous trouverez les réponses à toutes vos questions au sujet du multicasting sur Internet. Après une brève introduction sur cette problèmatique, Nous présenterons en exemple un panel non exhaustif de techniques et d'implémentations concrètes.

L'analyse des avantages et inconvénients de ces différentes technologies nous mènerons au protocole PIM SM (Protocol Independent Multicast in Sparse Mode). Nous étudierons le fonctionnement de ce dernier plus en profondeur pour enfin passer à la réalisation d'un simulateur.

L'objectif poursuivi lors de la réalisation du simulateur est purement didactique. Il n'est pas question d'implémenter une version fonctionnelle exécutable par un routeur du réseau. Le but est de réaliser une application qui puisse servir de support à la présentation du fonctionnement du protocole dans un réseau plus ou moins complexe, du rôle joué par chaque composant de ce réseau et des conséquences des choix effectués à la configuration de celui-ci.

mots clés: *Multicast, PIM, Protocol Independent Multicast, Sparse Mode, simulateur, réseau*

Abstract

This document is not a treatise in which you will find the answers to all your questions about the multicasting over the Internet. After a short introduction on these problems, We will introduce in example a panel of techniques and concrete implementations.

The analysis of the advantages and disadvantages of these various technologies we will lead to protocol PIM SM (Protocol Independent multicast in Sparse Mode). We will study the operation of this last one more in-depth for finally passing to the realization of a simulator.

The aim in view at the time of the realization of the simulator is purely didactic. It is not question to realize a functional implementation by a router of the network. The goal is to carry out an application which can be used as support to the presentation of the operation of the protocol in a more or less complex network, of the role played by each component of this network and the consequences of the choices carried out with the configuration of this one.

key words: Multicast, PIM, Protocol Independent multicast, Sparse Mode, simulator, network

Avant-propos

Je tiens à remercier :

Monsieur Olivier Bonaventure, promoteur de ce mémoire, pour sa disponibilité, son efficacité et ses conseils judicieux tout au long de ce travail.

L'ensemble des professeurs pour la formation de qualité qu'ils m'ont offerte.

Les membres du Jury pour l'intérêt qu'ils voudront bien porter à la lecture de ce travail.

Toutes les personnes qui, par leur aide et leur soutien, m'ont aidé à réaliser ce travail, en particulier ma mère.

Table des matières

Glossaire

Adresse IP : Adresse du Protocole Internet de couche réseau d'un dispositif. Une adresse IP est composée de 32 bits divisés en deux ou trois zones : un numéro de réseau (ou un numéro de réseau et un numéro de sous-réseau) et un numéro de station.

Adresse IP de classe D : Groupe d'adresses IP comprises entre 224.0.0.0 et 239.255.255.255 utilisées pour spécifier un groupe multicast.

Backbone : Ensemble des liaisons principales du réseau qui interconnecte l'ensemble des routeurs du réseau.

Broadcast : Méthode de transmission un pour tous (aucune gestion de groupe de receveurs)

Datagramme : Bloc indépendant contenant les données et adresses source et destination utilisées pour acheminer le paquet sur un domaine. Les datagrammes sont les unités d'information primaires utilisées sur l'Internet..

DM : (Dense Mode) Se dit d'un protocole de routage multicast prévu pour fonctionner dans un environnement possédant une grande bande passante et comprenant de nombreux receveurs.

Domaine : Ensemble de LAN interconnectés à l'aide d'une série de routeurs constituant le backbone.

Emetteur : Voir Source

Groupe multicast : Ensemble de receveur désirant recevoir un flux multicast référencé par une adresse IP de classe D

IETF : (Internet Engineering Task Force) Comité examinant et supportant les propositions relatives aux protocoles Internet.

IP : (Internet Protocol) Protocole de la couche réseau qui constitue la norme pour envoyer un datagramme à travers Internet

LAN : (Local Area Network) Réseau local regroupant un groupe de stations et possédant une adresse réseau unique.

Liaison Point à Point : Caractérise un moyen de communication physique ne reliant que deux dispositifs matériels (contrairement à un LAN qui est multipoint)

Membre : Voir Receveur

Multicast : méthode de diffusion simultanée des données en direction de plusieurs destinataires, sur un domaine.

PIM : (Protocol Independent Multicast) Protocole de routage des flux multicasts restant indépendant de toute méthode de routage unicast. PIM fonctionne en deux mode : DM et SM.

SM : (Sparse Mode) Se dit d'un protocole de routage multicast prévu pour fonctionner dans un environnement où il est nécessaire d'économiser la bande passante et comprenant un nombre limité de receveurs.

Receveur : Station connectée à un LAN désirant recevoir le flux multicast d'un groupe multicast référencé par une adresse IP de classe D

Routage : Méthode d'acheminement des informations à la bonne destination à travers un domaine. Selon les types de réseau, on envoie les données par paquets et on choisit leur chemin au coup par coup (C'est le cas pour Internet), ou bien on choisit un chemin une bonne fois pour toutes.

Routeur : Dispositif matériel ou logiciel permettant le routage des datagrammes vers le ou les bon(s) destinataire(s), dans un domaine.

Source : Station du domaine émettant des datagrammes à destination d'un groupe multicast référencé par une adresse IP de classe D

Unicast : Méthode de transmission un pour un (une source pour un receveur)

Introduction

Au début des années 1960, le département de la défense américaine a financé un projet de développement d'un protocole de communication basé sur la commutation de paquets. Ces recherches ont donné naissance au réseau ARPANET. En 1969, il reliait 4 ordinateurs. ARPANET était principalement utilisé par les communautés universitaires et les chercheurs.

Cependant, les possibilités offertes par le réseau ont rapidement intéressé la plupart des secteurs d'activités. Dès lors, ARPANET a évolué pour devenir Internet. Le nombre d'ordinateur connecté a suivi une croissance exponentielle. En 1983, Internet regroupait 200 ordinateurs. En 1995 on estime que 50000 réseaux étaient interconnectés. De nos jours, Internet est devenu incontournable. Plus de 25 millions de personnes ont accès au réseau des réseaux. On commence à parler des personnes qui ne savent pas utiliser le WEB en terme de "nouveaux illettrés".

Parallèlement à cette extension, Nous avons pu observer le développement du nombre de services offert sur Internet. Du simple échange de fichiers et d'applications de messagerie, nous sommes passés à l'ère du "World Wide Web" qui a révolutionné les méthodes de présentation de l'information. A l'aide de présentations de plus en plus complexes, interactives et attrayantes, Internet est devenu un média rivalisant avec la télévision, la radio et les imprimés.

Mais l'évolution d'Internet n'est pas terminée. Parallèlement (voir grâce) au développement des techniques multimédias, Internet est peut être en train de réaliser sa révolution la plus importante. Il ambitionne de devenir LE média et d'intégrer à ses services devenus traditionnels la radiodiffusion, la vidéodiffusion, la téléphonie, les jeux en ligne, la télé éducation, ...

Les possibilités, la demande et les enjeux sont énormes. Le seul frein reste la bande passante disponible et la gestion de celle-ci. L'extension quotidienne du backbone, tant en rapidité qu'en largeur de bande, demeure insuffisante pour supporter

l'augmentation de trafic engendré par le multimédia. Pour permettre cette dernière évolution, il est nécessaire d'ajouter à l'ensemble de protocoles TCP/IP un moyen de gérer efficacement les communications multicast. C'est ce que nous nous proposons d'étudier de plus près dans la suite de ce document.

Après un rapide passage en revue des différents modes de transmission, nous étudierons de plus en profondeur les outils permettant de mettre en œuvre une gestion du trafic multicast sur l'Internet. Nous passerons ensuite en revue un panel représentatif de méthodes permettant la gestion de ces flux. L'étude des avantages et inconvénients de ces différentes méthodes nous mènerons à une solution plus aboutie. C'est le protocole PIM SM.

Nous réaliserons une étude plus approfondie de celui-ci. A la suite de quoi, nous nous attaquerons à la réalisation du simulateur de ce protocole. Ce simulateur à but didactique permettra la représentation dynamique d'un domaine PIM SM, la configuration des différents éléments de ce domaine, l'échange des messages permettant la gestion du protocole et la diffusion des différents flux multicast actifs sur ce domaine.

1. Internet, c'est quoi ?

En réalité, Internet n'est pas un réseau au sens propre du terme. C'est plutôt une interconnexion d'un grand nombre de réseaux indépendants l'un de l'autre. Chacun d'entre eux utilise l'architecture qui correspond le mieux à ses besoins, ainsi qu'une politique qui lui est propre pour la gestion des communications entre les différents équipements qui le compose.

Ces différents réseaux sont reliés entre eux par un ensemble d'appareils que l'on appelle routeurs. Ces routeurs permettent d'ouvrir un réseau à l'Internet et de distribuer le flux de données en les dirigeant de leur source vers leur(s) destination(s) en respectant les principes de fonctionnement de l'Internet.

L'ensemble des routeurs permettant la connexion des différents réseaux forme ce que l'on appelle le BackBone. L'enjeu est de parvenir à regrouper tous ces groupes indépendants et de permettre aux informations de parcourir l'Internet en passant d'un réseau à l'autre sans se soucier de leur architecture. Pour y arriver, un ensemble de protocole a été développé (TCP, UDP, IP, adressage, protocoles de routage, ...). Aujourd'hui encore, cet ensemble évolue toujours en fonction des nouveaux besoins et des techniques.

Principes et avantages des différents modes d'émission

L'unicast

Depuis sa naissance et encore de nos jours les échanges sont principalement effectués en mode unicast. Ce type d'émission d'information permet à une machine source d'envoyer des datagrammes en direction d'une et d'une seule machine destinataire (cf. figure 1).

De nos jours, les progrès réalisés dans certains domaines (ex : le multimédia) nous font observer une augmentation de la popularité d'un nouveau type d'application tel

qu'audio et vidéo conférence. Ce type d'application met en œuvre une communication entre une source et plusieurs destinataires (one-to-many) ou entre plusieurs sources et plusieurs destinataires (many-to-many). Pour permettre à Internet de continuer son

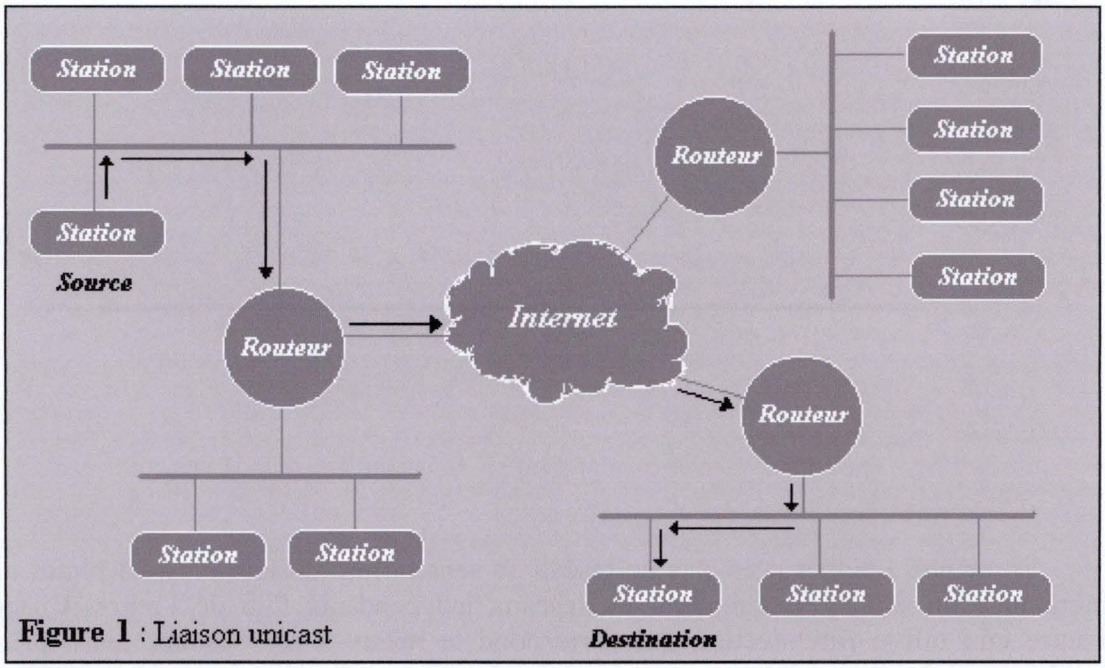


Figure 1 : Liaison unicast

extension, il est primordial de minimiser son engorgement et donc, de développer une nouvelle technique de transmission permettant une communication efficace pour ce genre de nouveaux besoins.

Le broadcast

Le broadcast (cf. figure 2) permet à une machine d'envoyer des paquets à toutes les machines en même temps. Les paquets contiennent l'adresse de la station source et

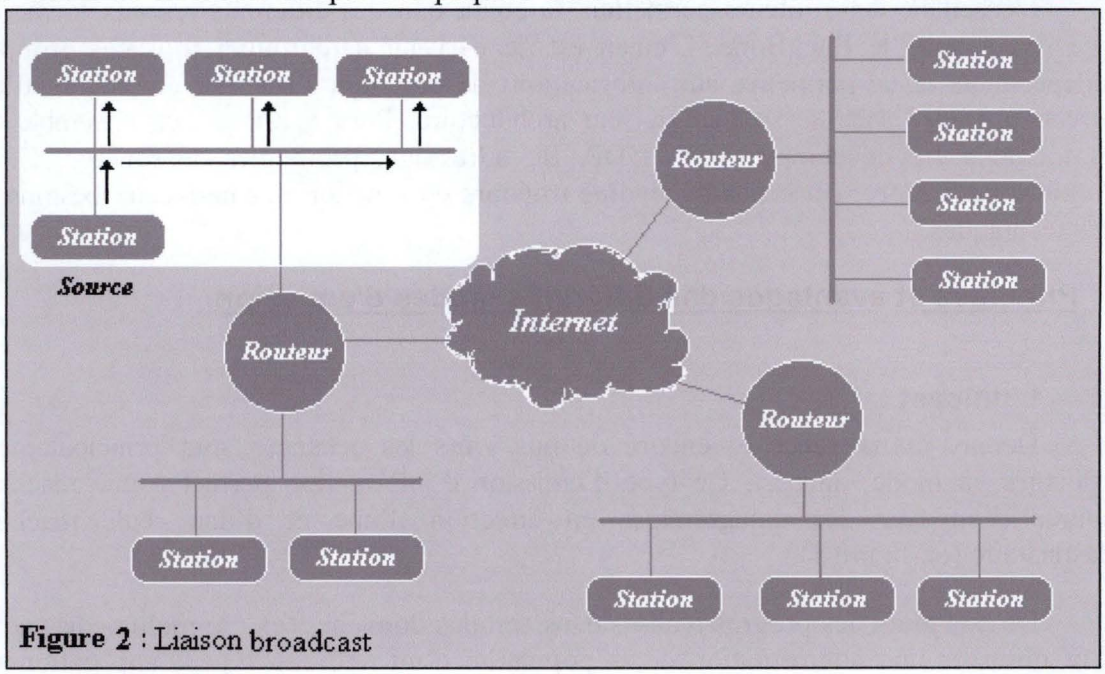


Figure 2 : Liaison broadcast

une adresse broadcast bien déterminée. Ce mode a l'avantage d'économiser la bande passante du réseau en n'émettant l'information qu'une seule fois et ce, quel que soit le nombre de stations intéressées par ces données. Il appartient aux stations d'accepter ou d'ignorer le paquet à son arrivée. Cependant, pour éviter que des paquets broadcast n'envahissent le monde entier, ceux-ci ne peuvent pas franchir les routeurs, ils restent confinés à leur LAN.

Le multicast

Pour le multicast (cf. figure 3), on ne s'adresse plus à une machine destinataire ou à une quelconque adresse broadcast. A la place, on émet à destination d'un groupe de machines. La notion de groupe est primordiale dans le multicast. C'est un ensemble de zéro ou plus de machines désigné par une adresse multicast.

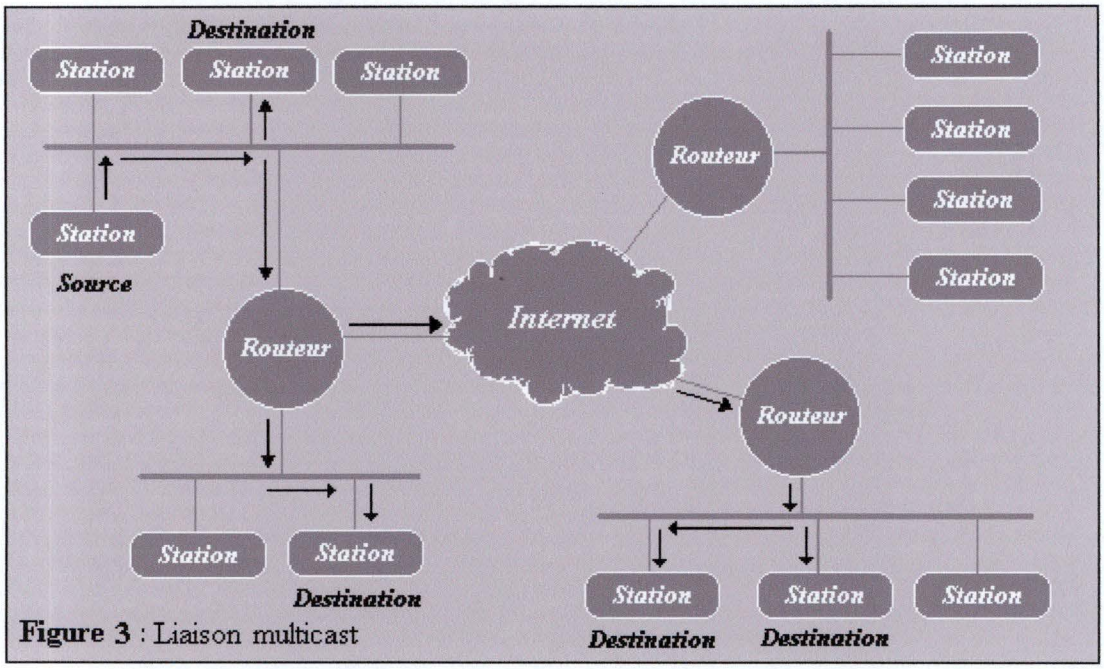


Figure 3 : Liaison multicast

Un groupe est géré entièrement dynamiquement. Les stations qui désirent recevoir les paquets destinés à un groupe doivent s'abonner à ce groupe et le quitter lorsqu'elles le désirent. La source, quant à elle, ne connaît que l 'adresse du groupe et absolument pas les adresses des stations qui y sont abonnées. Il n'y a aucune restriction sur le nombre de membres d'un groupe, sur leur localisation et sur le nombre de sources. Il est à remarquer qu'il n'est pas nécessaire d'être membre d'un groupe pour pouvoir en être une source et qu'une station peut être membre d'autant de groupes qu'elle désire.

Une bonne gestion du multicast par les routeurs permet de diffuser l'information vers tous les membres du groupe destinataire et uniquement eux en minimisant le flux de données. La même information ne passe au maximum qu'une fois sur chaque tronçon du réseau.

2. Le multicast sur Internet.

2.1. Adresses

Internet est géré par le protocole IP. Ce protocole définit toute une série de règles et entre autres, la structure des adresses. En IP version 4, les adresses comportent 32 bits (4 octets) et sont réparties en cinq classes (cf. figure 4)

- Classe A : de 1.0.0.0 à 126.255.255.255	<0xxxxxxx> . < XX . XX . XX >
- Classe B : de 128.0.0.0 à 191.255.255.255	<10xxxxxx . XX> . < XX . XX >
- Classe C : de 192.0.0.0 à 223.255.255.255	<110xxxxx . XX . XX > . < XX >
- Classe D : de 224.0.0.0 à 239.255.255.255	<1110 (+28 bits d'adressage groupe)>
- Classe E : réservées pour des expérimentations	

Figure 4 : les classes d'adresse IP ver 4

Pour organiser la distribution des adresses IP et pour éviter la duplication de l'une d'elles, un organisme a été créé. Cet organisme est l'IANA (Internet Assigned Numbers Authority). Les adresses de la classe D ont été réservées aux groupes multicast, ce qui représente près de 250 millions de possibilités.

De plus, l'IANA a réservé les adresses comprises entre 224.0.0.0 et 224.0.0.255 pour la gestion du routage. Par exemple :

- 224.0.0.1 : Toutes les machines multicast d'un LAN (portée locale)
- 224.0.0.2 : Tous les routeurs multicast d'un LAN(portée locale)
- 224.0.0.13 : Tous les routeur PIM

Les adresses comprises entre 224.0.1.0 et 238.255.255 sont réservées pour la gestion des groupes, certaines d'entre elles sont attribuées à un groupe bien précis. Par exemple :

- 224.0.1.11 : IETF-1-audio
- 224.0.1.12 : IETF-1-vidéo
- 224.0.19.0 à 63 : réservé à la Wald Disney Company

Pour finir, les adresses comprises entre 239.0.0.0 et 239.255.255.255 sont réservées à une utilisation locale. Les données ne doivent pas quitter le réseau d'origine pour être transférées sur l'Internet.

2.2. Protocole

Une grande partie des applications communiquant sur Internet passent par les services du protocole TCP (Transport Control Protocol). Celui-ci assure aux applications une connexion fiable bout à bout. TCP garantit que les datagrames arrivent à destination (en cas de perte de l'un d'entre eux, il s'occupe de sa retransmission) et qu'ils ne soient pas réordonnés.

Cette manière de procéder est très utile pour une application qui a besoin de la garantie que ses données arrivent bien à destination et dans l'ordre. Mais dans le cas du multicast, il n'y a pas de connexion entre 2 stations. La source ne connaît que l'adresse du groupe. De plus, pour des applications du type vidéoconférence, il est souvent plus intéressant de gagner le temps consommé par le protocole TCP pour la gestion de la connexion et des erreurs de transmission quitte à risquer d'obtenir quelques

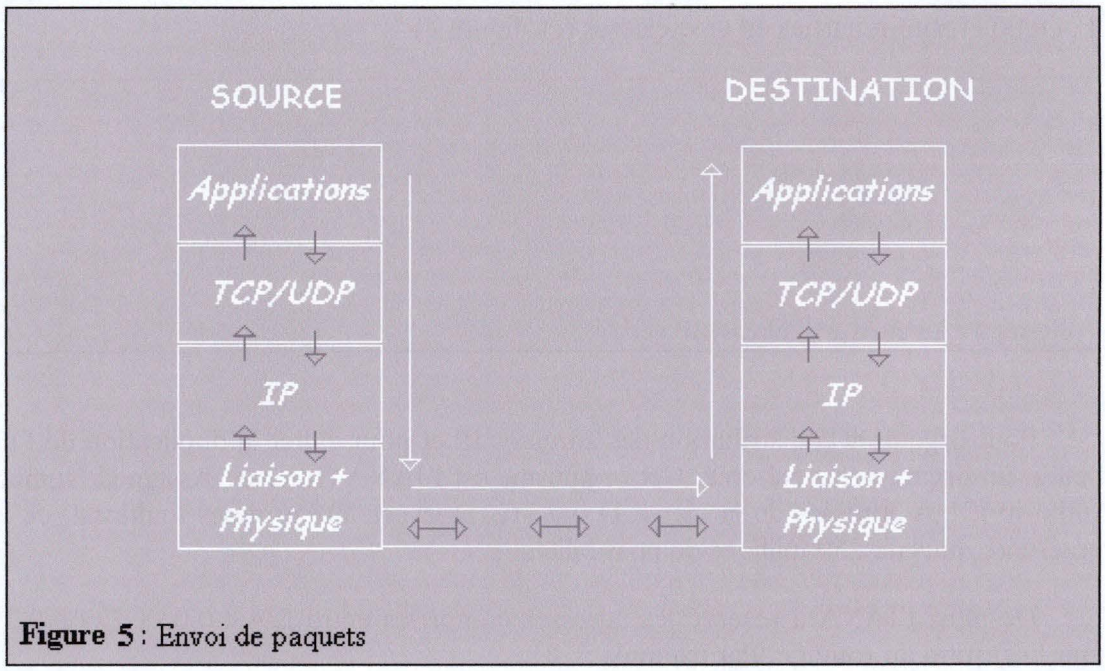


Figure 5 : Envoi de paquets

perturbations dans l'image ou le son. C'est pourquoi on utilise UDP qui laisse le soin à l'application de gérer la transmission comme bon lui semble.

2.3. Communication multicast sur un LAN

Comme illustré sur la figure 5, la couche TCP/IP (dans ce cas UDP + IP) est située au-dessus de la couche liaison de données. Celle-ci gère les échanges via le média physique Pour permettre une liaison entre deux applications via, les données sont transmises sur la station source du haut vers le bas, puis sur le media physique jusqu'au destinataire et enfin de bas en haut vers l'application du destinataire. Dans le cas d'une liaison unicast, on dispose d'un protocole (ARP) qui, à partir de l'adresse IP du destinataire, permet à la couche liaison de données de la station source de retrouver l'adresse de l'interface physique du destinataire. Ce mécanisme permet à la couche liaison de donnée d'encapsuler le datagram IP et de le faire parvenir à la couche IP du destinataire de façon transparente.

Dans le cas d'une communication multicast, on ne connait pas de l'adresse d'un destinataire, mais uniquement de l'adresse d'un groupe multicast. Un mécanisme tel que ARP n'est donc, dans ce cas-là, d'aucune utilité. Cependant, quelle que soit l'implémentation de la couche liaison de données choisie (Ethernet, Token Ring, ...), il est nécessaire de disposer d'un mécanisme permettant la diffusion du datagram sur le LAN via la gestion multicast propriétaire de celui-ci.

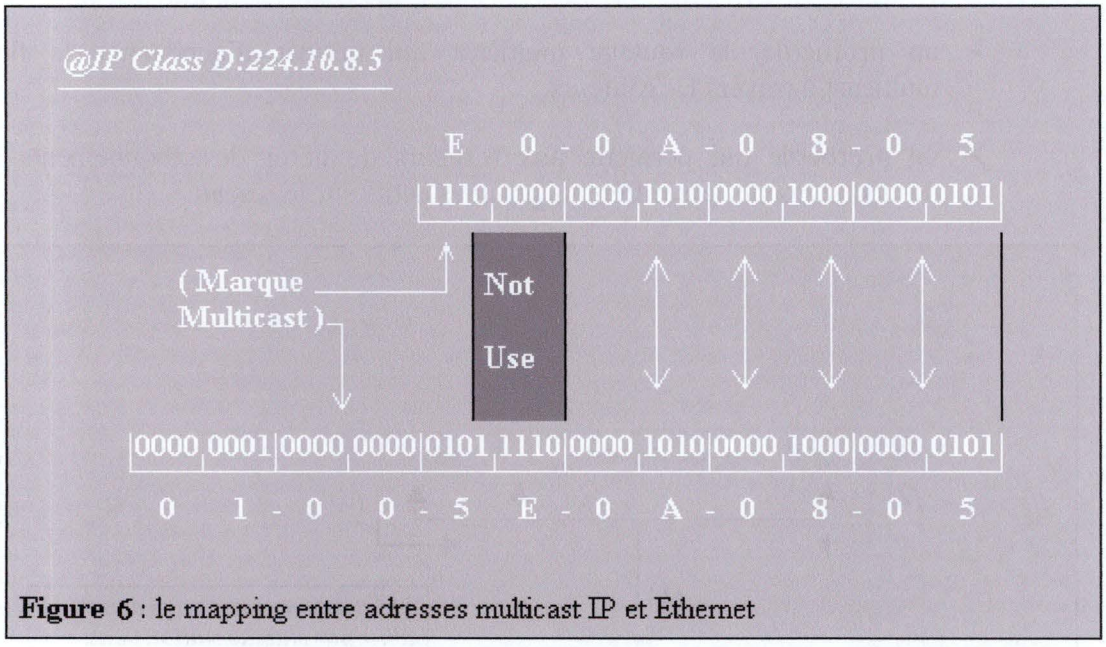


Figure 6 : le mapping entre adresses multicast IP et Ethernet

A simple titre d'illustration et sans entrer dans les détails, prenons le cas d'Ethernet (certainement le plus répandu). Les adresses TCP sont formées sur 32 bits alors que les adresses Ethernet en comptent 48. L'IANA a réservé les adresses Ethernet qui vont de 01.00.5E.00.00.00 à 01.00.5E.FF.FF.FF pour la gestion du multicasting. Lorsque l'on désire convertir une adresse multicast IP vers une adresse multicast Ethernet, il suffit de mapper les 24 bits de poids faible d'une adresse vers l'autre (cf. figure 6) à l'aide des masques 224.0.0.0 pour IP et 01.00.5E.00.00.00 pour Ethernet.

Il est intéressant de remarquer que ce procédé ne permet pas d'obtenir l'unicité. En effet, 32 (2^E5 c'est-à-dire les 5 bits de poids fort qui ne sont pas utilisés sur les 28 bits significatifs de l'adresse IP) groupes IP partagent la même adresse Ethernet. Par exemple, le groupe IP 224.10.8.5 (E0.0A.08.05) et le groupe IP 225.10.8.5 (E1.0A.08.05) sont tous deux mappés sur la même adresse Ethernet (00.5E.0A.08.05).

Pour ce qui est de la réception, la carte Ethernet du destinataire est, comme toutes les cartes Ethernet, configurée à la base pour écouter uniquement son adresse et l'adresse broadcast (FF.FF.FF.FF.FF.FF). Il est nécessaire de programmer explicitement les autres adresses que l'on désire également lui faire écouter. Pour le multicast, il est au minimum nécessaire de recevoir l'adresse Ethernet équivalente à l'adresse IP 224.0.0.1 (tous les hôtes multicast d'un LAN). Etant donné que la conversion des adresses ne donne pas l'unicité, l'interface Ethernet transférera à la couche IP les paquets destinés à tous les groupes mappés sur la même adresse. Il est donc nécessaire que la couche IP effectue un filtrage supplémentaire sur l'adresse IP du groupe.

2.4. Protocole IGMP

Dans le cas où les communications multicast restent confinées au même LAN, la procédure est relativement simple et la méthode décrite ci-dessus est suffisante. Cependant, lorsque l'on désire communiquer sur plusieurs LAN connectés entre eux par un ensemble de routeurs, les choses deviennent plus compliquées. Dans ce cas, deux protocoles seront nécessaires :

- un protocole de routage multicast qui permet d'acheminer le flux multicast à travers les routeurs.
- un protocole qui permette aux routeurs de gérer des abonnements et désabonnements aux différents groupes actifs sur le réseau.

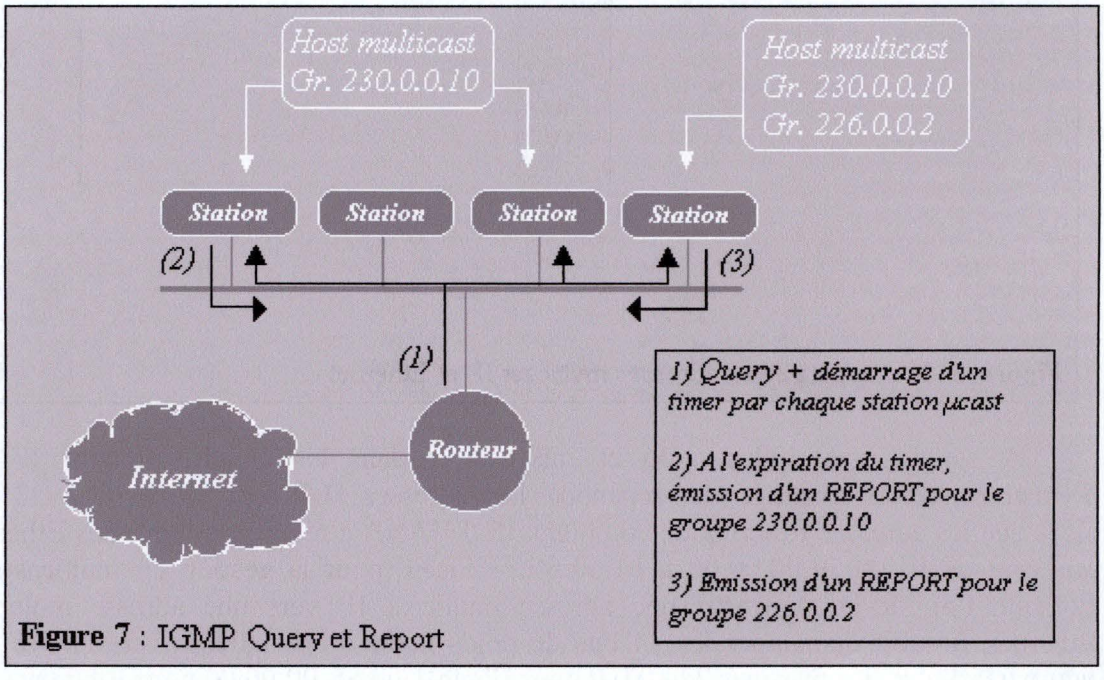


Figure 7 : IGMP Query et Report

Pour ce faire, on utilise l'Internet Group Management Protocol (IGMP). Ce protocole fonctionne entre les stations multicast d'un LAN et un routeur multicast directement connecté à celui-ci. Ce procédé permet aux routeurs multicast de connaître les groupes qu'ils doivent retransmettre sur le LAN. Si plusieurs routeurs multicast sont connectés au LAN, l'un d'entre eux est élu pour la gestion du protocole.

Le routeur désigné pour la gestion de l'IGMP envoie périodiquement (toutes les 60 secondes) une requête («QUERY ») à tous les hôtes multicast du LAN (Via l'adresse 224.0.0.1 avec un Time to live égal à 1). Cette requête signifie «A quel(s) groupe(s) voulez-vous être abonné ? ». A la réception du QUERY, les hôtes qui désirent recevoir l'un ou l'autre groupe répondent par un «IGMP REPORT» contenant les adresses les intéressant (cf. figure 7).

Pour éviter d'encombrer le réseau par des REPORTS contenant la même adresse, les stations attendent, à la réception d'un QUERY, un temps aléatoire avant de répondre. Au cours de cette période, si une autre station demande un groupe également désiré par celle-ci, il n'est pas nécessaire de le déclarer une deuxième fois. De plus, pour éviter une attente trop longue, lorsqu'une station désire joindre un groupe, elle envoie directement un REPORT sans attendre un QUERY.

IGMP version 2 apporte quelques améliorations à la première version. Entre autres, il définit une procédure qui permet à plusieurs routeurs multicast connectés à un LAN de choisir celui qui sera responsable de l'émission des QUERY. C'est en fait tout simplement celui qui possède la plus petite adresse IP. De plus, il fournit également un nouveau message permettant à une station de signaler directement un groupe qu'elle ne désire plus recevoir. Si la station en question était la dernière à recevoir ce groupe, le trafic sera ainsi diminué.

Dans la troisième version d'IGMP, il est possible à une station de choisir une source spécifique à l'intérieur d'un groupe. Cette amélioration permet de diminuer le trafic sur le LAN et également sur le backbone multicast.

2.5. Acheminement multicast à travers les routeurs

Cette partie abordera les principales techniques permettant aux routeurs de gérer le flux multicast sur le backbone.

2.5.1. Techniques simples

2.5.1.1. Inondation («Flooding »)

Il s'agit de la méthode la plus simple qui permette de délivrer le flux multicast à tous les routeurs du réseau. Lorsqu'un routeur reçoit un paquet adressé à un groupe multicast, celui-ci détermine s'il s'agit de la première fois que ce paquet lui parvient. Dans l'affirmative, il le retransmet sur toutes ses interfaces à l'exception de celle par où le paquet est arrivé. Dans le cas contraire, le paquet est simplement supprimé ce qui permet d'éviter les boucles.

Cette technique garantit la distribution de flux multicast à l'entièreté du réseau. Il est très simple d'en réaliser l'implémentation car les routeurs ne doivent pas gérer des tables de routage multicast. Ils doivent uniquement garder la trace de paquets récemment traités. Cependant, la gestion du flux n'est absolument pas optimisée. Ce manque d'optimisation entraîne une augmentation significative de l'occupation de la bande passante et rend cette technique inadéquate au niveau de l'Internet.

2.5.1.2. Spanning Trees

Un spaning tree est un arbre dans lequel tous les routeurs du réseau sont reliés par une et une seule route (cf. figure 9). L'idée est de faire gérer une telle structure par l'ensemble des routeurs. Ensuite, lorsqu'un paquet multicast arrive, il suffit de le renvoyer sur toutes les interfaces actives de cet arbre, exceptée celle par ou le message est entré.

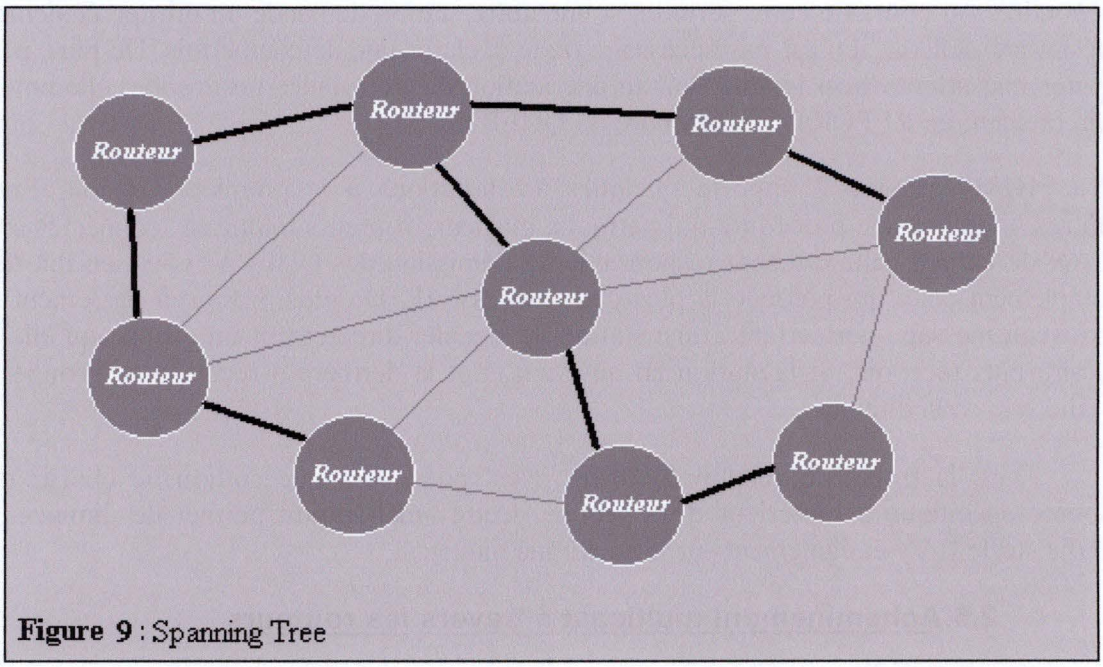


Figure 9 : Spanning Tree

Cette solution est efficace et relativement simple à implémenter. Cependant, elle centralise tout le trafic multicast sur le même ensemble de lignes. Ceci n'étant pas souhaitable sur Internet dans la mesure où elle entraîne, si le flux est important, un risque conséquent de surcharge des lignes composant cet arbre.

2.5.2. Techniques utilisant un arbre de diffusion par source

2.5.2.1. Reverse Path Broadcasting (RPB)

Au lieu de construire un spanning tree unique pour tout le réseau, une solution plus efficace consiste à construire un arbre à partir de chaque source active. Chaque source est donc la racine de son arbre de diffusion multicast.

L'algorithme utilisé pour gérer cette technique est le RPB. A l'arrivée d'un paquet multicast, le routeur l'examine. Si le paquet est arrivé par l'interface menant à la source par le plus court chemin (Reverse Path), il sera diffusé sur toutes les interfaces excepté celle d'entrée. Sinon, il sera tout simplement supprimé. Chaque routeur doit donc connaître la topologie du réseau et en particulier, les interfaces qui mènent, par le chemin le plus court, à chacune des sources potentielles.

Cette technique est relativement simple et efficace tout en ne nécessitant que peu de ressources aux routeurs. De plus, elle garantit que les paquets seront délivrés par le chemin le plus court tout en évitant d'utiliser toujours les mêmes liaisons pour l'ensemble du flux multicast.

2.5.2.2. Truncates Reverse Path Broadcasting (TRPB)

Si, à la technique précédente (RPB), on ajoute la prise en charge du protocole IGMP entre les routeurs et les LAN qui y sont directement connectés, il devient possible aux routeurs de déterminer les groupes demandés par les stations d'un LAN. Ils peuvent ainsi filtrer le trafic multicast et ne laisser entrer sur un LAN que les paquets attendus. C'est ce qui est réalisé par le TRPB.

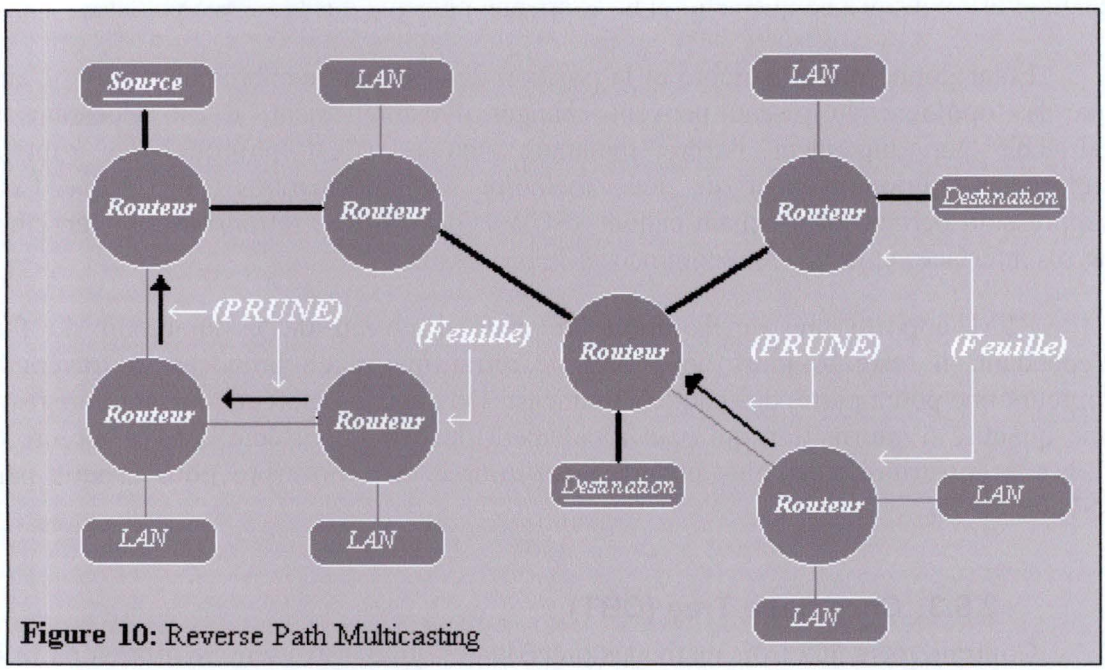


Figure 10: Reverse Path Multicasting

TRPB permet donc de limiter le trafic multicast transmis sur les différents LAN du réseau. Cependant, sur le backbone, la liaison est toujours de type broadcast. Il reste toujours du trafic inutile traversant certaines liaisons entre routeurs.

2.5.2.3. Reverse Path Multicasting (RPM)

Reverse Path Multicasting permet de supprimer le trafic inutile entre routeurs. Pour cela, on améliore le TRPB. Non seulement les routeurs ne transmettent que le trafic multicast nécessaire sur un LAN, mais également entre eux. Le flux multicast

n'est plus broadcasté entre la totalité des routeurs, mais uniquement sur les lignes nécessaires pour l'acheminer aux stations membres du groupe.

Pour ce faire, à l'arrivée d'un paquet «SRC→GRP» (c'est-à-dire en provenance de la source SRC et adressé au groupe multicast GRP) et s'il s'agit du premier paquet de ce flux, il est retransmis à tous les routeurs en suivant l'algorithme TRPB. Par ce principe, tous les routeurs «feuilles » (situés au bout d'une branche de l'arbre TRPB) ont la garantie de recevoir le premier paquet d'une communication multicast.

Sur base des informations fournies par IGMP, le routeur peut transmettre le flux à un LAN qui lui est directement connecté si une station le désire. Si aucun de ces LAN ne demande la réception de ce flux, il prévient le routeur précédent qu'il est inutile de lui transmettre ce flux. Pour cela, il lui envoie un message contenant l'identification «SRC→GRP» du flux inutile. Ce message porte le nom de «PRUNE».

Lorsqu'un routeur réceptionne un message «PRUNE», il enregistre dans sa mémoire le fait qu'il ne doit plus retransmettre le flux «SCR→GRP» sur l'interface d'entrée du «PRUNE». Dans le cas où il ne doit plus retransmettre ce flux sur aucune de ses interfaces, le routeur transmet à son tour un message «PRUNE» pour ce flux au routeur directement précédent. Petit à petit, cette succession de message «PRUNE» élague l'arbre jusqu'à ce que celui-ci ne contienne plus que des branches vivantes.

Etant donné que le nombre et la position des stations membres d'un groupe ainsi que la topologie du réseau peuvent changer dynamiquement, il est nécessaire de rafraîchir périodiquement l'arbre minimum obtenu. C'est pourquoi les routeurs suppriment périodiquement de leur mémoire les informations «PRUNE». Cette suppression permet au prochain paquet «SCR→GRP» d'être retransmis sur l'ensemble de ses interfaces et donc, de recommencer le processus.

RPM apporte une amélioration certaine au niveau de la limitation du flux. Cependant, il reste toujours nécessaire de retransmettre en broadcast à travers les routeurs une petite partie des paquets multicast. De plus, les routeurs doivent enregistrer une quantité d'information qui peut rapidement devenir importante dans la mesure où l'arbre à construire n'est plus un arbre par source mais un arbre pour chaque paire «SCR→GRP».

2.5.3. Core Base Tree (CBT)

Contrairement aux trois méthodes précédentes qui construisent un arbre à partir de chaque source (voir même chaque paire «SCR→GRP») ou au spanning tree qui construit un arbre unique pour tout le trafic multicast, cette méthode construit un arbre unique pour chaque groupe. La totalité du flux d'un groupe utilise le même arbre pour se répandre sur le réseau et ce, quelle que soit la source (cf. figure11).

Pour construire l'arbre, un ou plusieurs routeurs sont élus pour servir de base à l'arbre du groupe. Ces routeurs faisant office de points de jonctions sont appelés «CORE». Lorsqu'une station désire joindre un groupe multicast, elle fait parvenir un message unicast à un des routeurs «CORE» du groupe. Ce message est nommé «JOIN» et contient l'adresse du groupe désiré par le nouveau destinataire. La station doit donc

connaître l'adresse IP de ce routeur. Le message «JOIN» est transféré de routeur en routeur sur le chemin qui va de la station au routeur «CORE».

Chacun des routeurs intermédiaires analyse le message «JOIN» et enregistre le fait que l'interface d'entrée de celui-ci fait maintenant partie de l'arbre multicast du groupe demandé. Lorsque le message arrive au routeur «CORE», l'arbre est complété et peut diffuser le flux jusqu'à la station.

Comme pour les autres techniques, une station source ne doit pas obligatoirement être membre du groupe. Dans ce cas, le flux émis par cette station est simplement encapsulé dans des paquets unicast. Ceux-ci sont expédiés à l'adresse du routeur «CORE» du groupe. Lorsqu'ils arrivent au premier routeur faisant partie de l'arbre multicast du groupe, celui-ci les intercepte et retransmet le flux en multicast.

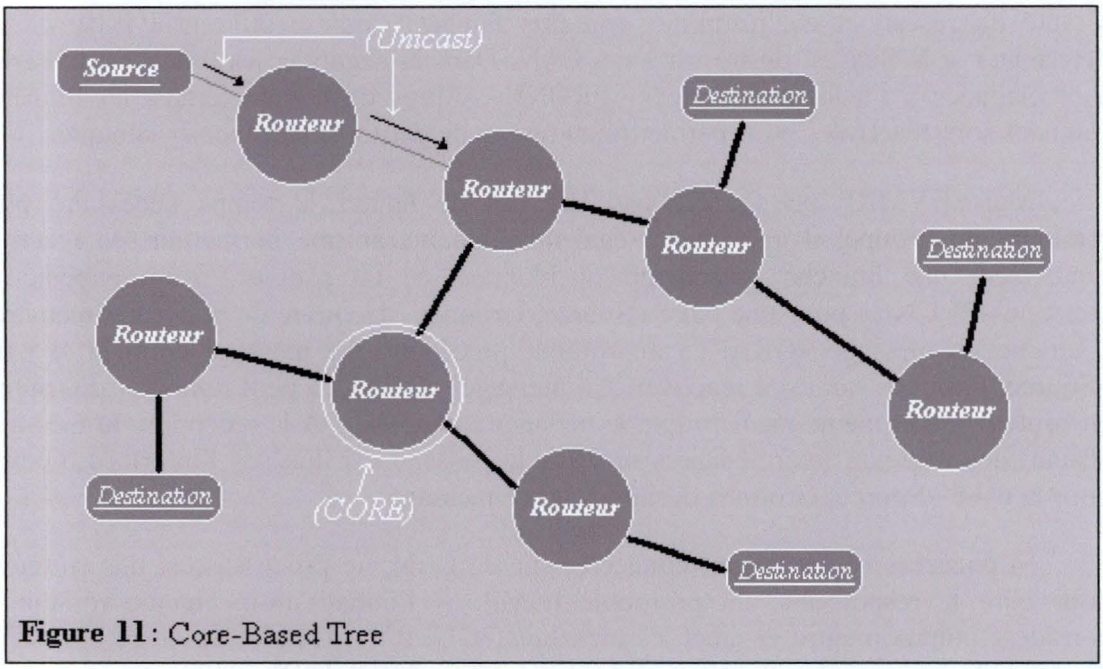


Figure 11: Core-Based Tree

De cette manière, le flux multicast n'est plus retransmis que sur les lignes nécessaires en éliminant le surplus d'émission en broadcast que l'on retrouve dans la méthode RPM.

Cependant, il reste quelques problèmes dont il faut tenir compte. Premièrement, les routes (Source→Destination) ne sont pas optimisées ce qui peut entraîner une augmentation du temps minimum nécessaire à la transmission des paquets. Deuxièmement, lorsque le traffic d'un groupe devient important, les lignes des routeurs «CORE» risquent d'être surchargées. Enfin, il est nécessaire de développer de nouveaux algorithmes pour la gestion de ces points de jonctions.

2.6. Implémentations concrètes de ces techniques

Il existe à l'heure actuelle un certain nombre d'implémentations de ces méthodes de routage multicast. Chacune d'entre elles apporte ses avantages et ses inconvénients qui les prédisposent pour tel ou tel type d'utilisation. Le but de ce travail n'est pas

l'étude de toutes ces méthodes. Cependant, il est intéressant de se pencher sur 2 d'entre elles.

2.6.1. Protocole DVMRP

Le protocole DVMRP (Distance Vector Multicast Routing) est intéressant car il est à l'origine du Mbone. Comme son nom l'indique, il utilise une méthode de routage par vecteur de distance et ce, afin de déterminer le chemin le plus court vers la source. A l'origine, il était dérivé de la méthode TRPB ce qui nécessitait la gestion d'un arbre par source. Cependant, à partir de la version 3, DVMRP utilise une implémentation de RPM. Cette particularité entraîne la nécessité de gérer un arbre par paire «Source, Groupe».

Comme prévu par RPM, le premier datagram est transmis en broadcast sur la totalité du réseau et ce, jusqu'aux routeurs feuilles. Ceux-ci utilisent IGMP afin de déterminer si le flux est désiré sur leurs LAN. Dans la négative, les branches de l'arbre sont élaguées à l'aide de messages «PRUNE». Après un certain temps, les branches élaguées sont réactivées pour permettre au réseau de garder son aspect dynamique.

Mais DVMRP ne s'arrête pas là. Afin de limiter le temps nécessaire pour rejoindre un groupe, il implémente également un mécanisme permettant de réactiver rapidement une branche précédemment bloquée. Si un routeur, ayant expédié un message «PRUNE» pour une paire «Source, Groupe», découvre de nouveaux membres, il envoie un message «GRAFT» au routeur précédent. Ce message contient la paire «Source, Groupe» qui est à réactiver. Un message «GRAFT» peut remonter de routeur en routeur, de la même manière que le message «PRUNE». A la réception du message, chacun des routeurs intermédiaires réactive le passage du flux sur l'interface d'entrée pour la paire «Source, Groupe» désignée par ce message.

Si plusieurs routeurs sont connectés sur un LAN, un processus est mis en œuvre pour élire le responsable du protocole IGMP. A l'initialisation, chaque routeur se considère comme maître et émet les messages «QUERY». Il ne cesse de se considérer responsable qu'au moment où il reçoit un message «QUERY» en provenance d'un routeur possédant une adresse IP inférieure à la sienne. De plus, pour éviter la duplication de paquets sur un LAN par plusieurs routeurs connectés à une même source, un de ces routeurs est élu dominant pour cette source: celui qui est relié à la source par le chemin le plus court et, en cas d'égalité, le routeur dont l'adresse IP est la plus petite.

Il reste à aborder un des points essentiels: comment décider des routes ? Un routeur DVMRP ne retransmet un paquet multicast que si celui-ci entre par l'interface qui mène à la source par le chemin le plus court (c'est-à-dire par le chemin où la somme des métriques de chaque tronçon est la plus petite). Un routeur DVMRP a donc besoin de garder en mémoire la liste des sources potentielles ainsi que l'interface qui y mène par le chemin le plus court. En réalité, il n'est pas nécessaire de traiter chacune des sources possibles mais plus simplement chacun des sous réseaux possibles. Pour chacun des sous réseaux, la table de routage doit donc contenir

- ✓ l'adresse IP du sous réseau,
- ✓ la métrique du chemin le plus court,

✓ l'interface qui y mène par le chemin le plus court.

Les routeurs DVMRP doivent périodiquement s'échanger leur table de routage afin que chacun puisse connaître la topologie complète du réseau. Au départ, un routeur ne connaît que les routes auxquels il est directement connecté. Au fur et à mesure du processus d'échange des tables, il découvre la structure du réseau et adapte sa propre table en fonction des informations fournies par les autres routeurs (cf. figure 13).

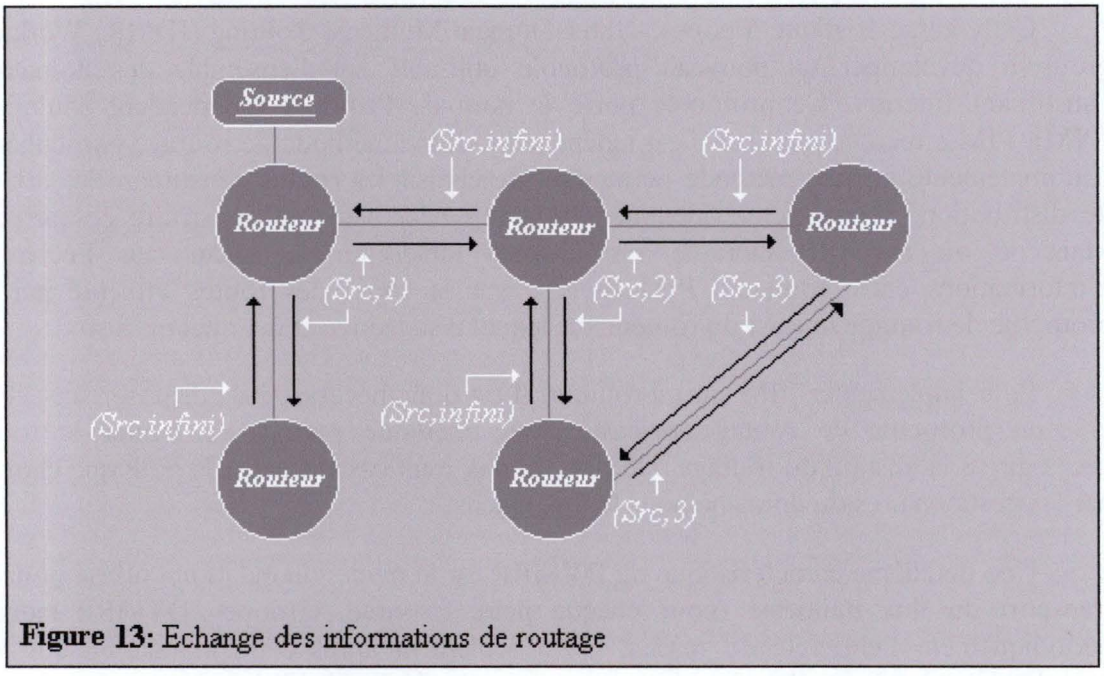


Figure 13: Echange des informations de routage

De plus, DVMRP utilise la technique de «Poison Reverse» : un routeur annonce une route de poids infini au routeur qu'il estime être à l'origine du chemin le plus court vers une source. Le routeur qui réceptionne une table de routage d'un routeur voisin avec une information de poids infini pour telle source sait qu'il lui appartient d'étendre l'arbre en le faisant passer par ce routeur voisin.

La table ainsi obtenue tient compte de toutes les sources possibles. Cependant, elle ne tient aucun compte de la gestion des groupes. En effet, une station peut être une source pour plusieurs groupes. En fonction des différents membres de chacun de ces groupes, un arbre différent doit être géré par paire «Source, Groupe». Un routeur DVMRP doit donc gérer une deuxième table qui, pour chacune des sources possibles, contient les informations nécessaires à la gestion du flux multicast pour chaque groupe actif à partir de cette source, à savoir:

- ✓ l'adresse du groupe,
- ✓ les interfaces réellement actives (interfaces pour lesquelles un message «PRUNE» n'est pas arrivé ou que celui-ci a été supprimer par l'arrivée d'un «GRAFT»).

2.6.2. Protocole PIM

L'étude de DVMRP permet de dégager une caractéristique importante sur son mode de gestion du routage : DVMRP implémente son propre protocole de routage. Cette méthode de travail entraîne une augmentation du trafic du au besoin d'échanger les informations de routage pour le flux multicast en plus des informations pour le flux unicast. Elle oblige les routeurs à consommer de la mémoire supplémentaire pour la gestion de cette table. De plus, il n'y a aucune préoccupation des choix éffectués par les gestionnaire du domaine pour la méthode le routage unicast.

Cette caractéristique a poussé l'Inter-Domain Multicast Routing (IDMR) Working group à développer un nouveau protocole utilisable sur l'ensemble des domaines constituant Internet. Ce protocole porte le nom de Protocol Independent Multicast (PIM). PIM a reçu ce nom car il est indépendant d'une méthode de routage particulière. Il n'implémente aucune méthode permettant de choisir les routes constituant les arbres de distribution. Il est bien évidemment toujours nécessaire de construire ces arbres. Mais, là où DVMRP fabrique ses propres tables en se basant sur l'échange d'informations entre routeurs, PIM se base sur le choix des routes effectué par le protocole de routage unicast du routeur sur lequel il se trouve et ce, quel qu'il soit.

Pour implémenter PIM sur un routeur, il est donc nécessaire de disposer sur celui-ci d'un protocole de routage unicast. Cette méthode permet de limiter le trafic nécessaire à la gestion du routage. De plus, le flux multicast respecte la politique choisie par les gestionnaires du domaine pour le flux unicast.

Une deuxième caractéristique de DVMRP est le mode «inondation» utilisé pour le transport du flux multicast (pour chaque paire «Source, Groupe», DVMRP inonde périodiquement l'entièreté du réseau. C'est seulement après cette inondation que les branches inutiles sont élaguées pour permettre de limiter le flux). Cette méthode est performante pour la gestion de groupes fortement représentés (Dense Mode). Cependant, pour la gestion de groupes faiblement représentés et largement dispersés (Sparse Mode), l'inondation périodique présente un risque important de déclencher des problèmes de performance.

Compte-tenu de leur utilité, IDMR a conservé ces deux méthodes. PIM effectue la distinction entre une méthode routage désignée pour travailler dans un environnement dense (PIM Dense Mode) et une autre pour travailler dans un environnement éparpillé (PIM Sparse Mode).

PIM Dense Mode est désigné pour travailler dans un environnement où les membres des groupes multicast sont concentrés dans une région possédant une bande passante relativement importante. Le réseau d'un campus universitaire est l'un des exemples les plus approprié pour l'utilisation de ce mode.

PIM Dense Mode est semblable au protocole DVMRP. C'est-à-dire qu'en dehors du fait de ne pas gérer le choix de routes (PIM Dense Mode se fie au protocole de routage unicast), il utilise également l'algorithme RPM pour la gestion du flux multicast. De plus, il implémente également l'utilisation de messages «GRAFT» pour réactiver une branche précédemment bloquée par un message «PRUNE».

La majorité des communications dans l'Internet est généralement moins concentrée que dans un campus universitaire et la bande passante est, en général, une denrée précieuse qu'il convient d'économiser au maximum. C'est pour ces raisons que le mode éparse de PIM (PIM Sparse Mode) a été conçu. Contrairement au mode dense, il n'y a pas dans ce cas d'inondation périodique de tout le réseau. Dans la suite de ce travail, nous étudierons plus en détail son mode de fonctionnement.

3. PIM Sparse Mode

3.1. Les spécificités du protocole

Les algorithmes tel que DVMRP ou PIM DM sont efficaces lorsqu'ils fonctionnent dans un environnement où les récepteurs multicast sont concentrés et disposent d'une bande passante relativement large. Dans le cas contraire, l'inondation périodique qu'ils nécessitent ne les rend pas très rentables.

C'est cette constatation qui a poussé l'IDMR working group à développer un protocole performant pour gérer le flux multicast issu de nombreuses sessions concurrentes dans un environnement épars: PIM SM. Mais détaillons quelque peu les spécificités de ce protocole :

- ✓ **Mode épars** : La principale particularité de PIM SM est son habilité à gérer le trafic multicast dans un milieu épars, c'est-à-dire dans un domaine où les réseaux contenant des receveurs sont significativement moins nombreux que les réseaux sans receveurs.

- ✓ **Qualité de diffusion** : PIM SM est capable de switcher du mode basé sur un arbre de distribution multicast partagé (comme pour CBT) avec pour racine le point de rendez-vous du groupe à un arbre de diffusion à chemin le plus court (avec la source comme racine). Cette particularité permet d'optimiser le comportement en fonction du besoin de chaque groupe et de contourner le risque d'engorgement au point de rendez-vous.

- ✓ **Indépendant d'une méthode de routage** : PIM est indépendant d'une méthode de routage par le fait qu'il n'en implémente aucune. A la place, il utilise les services de la méthode de routage unicast, ce qui permet d'unifier la gestion des deux simplifiant ainsi les problèmes de déploiement et de gestion.

- ✓ **Résistant aux changements de topologie du réseau** : Outre la simplification de la tâche des gestionnaires du réseau, le recours à la technique de routage unicast permet de profiter des qualités de cette dernière. Ces techniques de routage unicast s'adaptent automatiquement aux besoins réels des réseaux et à leurs changements de topologie.

3.2. Les éléments constitutifs d'un domaine PIM SM

Comme tout protocole, PIM SM possède son propre jeu d'éléments, dont les principaux sont les suivants :

- ✓ **Membre** : Un membre est une station désirant recevoir le flux multicast d'un ou de plusieurs groupe(s). Un membre est également référencé comme "receveur".

- ✓ **Source** : Une source est une station qui émet du trafic en direction d'un groupe multicast. Une source est également référencée comme "émetteur".

- ✓ **Designated router (DR)** : Dans un domaine multicast, les LAN peuvent être directement connectés à plusieurs routeurs PIM SM. Dans une telle configuration, un seul de ces routeurs est désigné pour gérer le trafic multicast des sources et des receveurs directement connectés à ce LAN. CE DR. est responsable de la gestion du protocole IGMP, de l'acheminement des différents flux demandés et de la retransmission des flux issus de sources directement connectées à ce LAN. Lorsque plusieurs routeurs sont connectés à un même LAN, c'est le routeur possédant l'adresse IP la plus élevée qui est élu responsable.

- ✓ **Point de rendez-vous (RP)** : Un point de rendez-vous (RP) est un routeur du domaine multicast élu pour servir de racine à l'arbre partagé (RT-tree) d'un groupe ou de plusieurs groupes multicasts.

- ✓ **Shared tree (RP-tree)** : Un shared tree (arbre partagé) est l'arbre de routage qui supporte le trafic d'un groupe multicast dont la racine est le RP du groupe. Son architecture est semblable à l'arbre de routage de la méthode CBT dans la mesure où il permet de faire passer l'ensemble du trafic émis par toutes les sources d'un groupe multicast.

- ✓ **Shortest Path tree (SP-tree)** : Contrairement au RP-tree, le Shortest Path tree a pour racine le DR d'une source. Celui-ci relie les membres du groupe à une source par le chemin le plus court. Il n'est donc plus question d'un arbre par groupe, mais d'un arbre par paire "Source-Groupe".

- ✓ **BootStrap Router (BSR)** : Le BootSrap Router est le routeur offrant les fonctionnalités permettant l'identification des RP pour les différents groupes multicast.

- ✓ **Candidate BSR (CBSR)** : Un candidat BSR est un routeur qui peut jouer le rôle de BSR pour le domaine multicast.

- ✓ **Candidat RP (CRP)** : Un candidat RP est un routeur qui peut jouer le rôle de RP pour un ou plusieurs groupe(s) multicast du domaine.

3.3. Le protocole en action

Comme tous les protocoles de routage évolués, PIM SM est un protocole distribué. C'est à dire qu'il définit les relations existant entre les différents éléments du réseau permettant à ceux-ci d'atteindre l'objectif fixé. Dans le cas de PIM SM, cet objectif est un routage efficace du flux multicast dans un réseau à topologie variable.

Le deuxième aspect intéressant pourrait être l'analyse des besoins permettant l'implémentation du protocole dans un routeur. Bien que n'étant pas l'objectif de ce travail, afin de faciliter la compréhension du protocole, il est intéressant d'aborder la gestion des informations par ces routeurs. Pour mener à bien sa tâche, un routeur PIM SM doit nécessairement tenir à jour un ensemble d'informations concernant la configuration du réseau multicast et des arbres de diffusion des flux actifs passant par celui-ci. Ces informations sont contenues dans une table : la base d'information des arbres (TIB). Cette TIB est essentiellement construite par le routeur sur base des messages IGMP qu'il reçoit d'un LAN pour lequel il œuvre en temps que DR et des JOIN, PRUNE et autre message PIM SM en provenance d'autres routeurs multicast du domaine.

Quels que soient les choix d'implémentation effectués pour l'interrogation et la mise à jour de cette table (chaque constructeur étant libre d'effectuer ses propres choix), elle doit contenir un minimum d'informations concernant le domaine multicast. On y retrouve entre autre les renseignements concernant le BSR du domaine, les routeurs PIM SM voisins ainsi que ses activités en tant que DR. La TIB doit également contenir les informations concernant les groupes multicast actifs sur ce routeur. Pour chacun de ses groupes, la TIB est répartie en trois sections (Plus de détails sur la gestion de cette TIB seront fournis dans la suite.) :

1. **La section (*,G):** Elle regroupe les informations concernant la situation de l'arbre partagé du groupe. Outre les différents timers nécessaires à la gestion de ce flux, on y retrouve les informations concernant la structure de l'arbre et principalement la liste des interfaces par lesquels le routeur doit retransmettre le flux (*,G) à l'intention des routeurs voisins. Dans le cas d'un DR, on y trouve également la liste des interfaces directement connectées à un LAN pour lequel il œuvre en temps que DR et ayant au moins un receveur pour ce groupe.

2. **La section (S,G):** Elle regroupe pour chaque source S les informations concernant l'état de l'arbre du groupe G spécifique à cette source (SP-tree). On y retrouve plus ou moins les mêmes informations que dans la section (*,G).

3. **La section (S,G,rpt) :** Cette dernière section regroupe les informations spécifiques à la source S nécessaire à la gestion du RP-tree.

Il est également nécessaire de gérer une série d'informations de routage des flux multicast qui sont reprises dans la base d'informations de routage multicast (MRIB). Cette MRIB permet essentiellement de retrouver l'identification du routeur PIM SM voisin menant à une source S ou au RP d'un groupe G. Vu la principale caractéristique de PIM SM (Protocol Independant), la MRIB est construite à partir des informations fournies par le protocole de routage unicast du routeur.

3.3.1. La répartition des rôles

Afin de permettre le bon fonctionnement d'un domaine PIM SM, il est nécessaire de définir le rôle de chacun de ses routeurs. La première solution est de configurer manuellement tous les routeurs du domaine. Cependant, cette manière de procéder peut être longue et source de nombreuses erreurs de configuration. Mais, plus important, elle limite la flexibilité du réseau. Le protocole ne pouvant pas s'adapter facilement aux changements de topologie de celui-ci (par exemple: Quid si un RP tombe en panne ?). Pour ces raisons, une série de techniques automatiques a été mise au point. Cette étape de bootsraping permet l'élection d'un BSR parmi les différents CBSR s'étant fait connaître au domaine, l'identification des RP potentiels, l'identification des routeurs PIM SM voisins et l'identification des DR de chaque LAN du domaine.

Dans un premier temps, les CBSR transmettent aux routeurs du domaine un message confirmant leur aptitude à œuvrer comme BSR. Le message transmit contient l'adresse IP du routeur ainsi qu'une valeur de préférence qui est préconfigurée dans le routeur. Le BSR élu est le CBSR possédant la valeur de préférence la plus élevée. En cas d'égalité de valeur de préférence, le routeur possédant la plus grande adresse IP est choisi.

Lorsqu'un routeur commence à travailler en temps que BSR, il est nécessaire qu'il collecte la liste de RP potentiels. Pour ce faire, les candidats RP du domaine émettent un message stipulant leur aptitude à travailler en temps que RP. LE BSR recueille ces messages et gère une liste contenant les adresses des RP potentiels qu'il distribue aux différents DR du domaine. Lorsqu'un nouveau groupe multicast est nécessaire, le DR applique une fonction de hashing sur les adresses IP de cette liste afin d'identifier le RP pour ce groupe.

Il est encore nécessaire d'élire les différents DR. Sur chaque LAN du domaine multicast un et un seul DR doit être actif. Lorsqu'un LAN est raccordé au domaine par plusieurs routeurs, il est nécessaire d'élire l'un de ces routeurs pour œuvrer en qualité de DR pour ce LAN. Les différents routeurs connecté à un LAN se font connaître par l'intermédiaire du message 'HELLO'. Celui-ci est émis par chaque routeurs PIM sur chacune de ses interfaces et a pour but de faire connaître le routeur à tous ses voisins et ce, que l'interface soit connectée à une ligne 'point-à-point' ou à un LAN. Tous les routeurs d'un LAN réceptionne donc le message 'Hello' émis par chaque routeur connecté à ce LAN. Ces messages contiennent l'adresse IP de leur émetteur et, dans cette situation, c'est simplement le routeur possédant la plus grande adresse IP qui est élu DR pour ce LAN.

Sur chaque LAN, il est également nécessaire de définir le routeur responsable de l'acheminement d'un flux multicast et ce, pour chaque flux actif sur ce LAN. Cette opération est réalisée par l'intermédiaire de messages '(S,G) ASSERT' et '(*,G) ASSERT'. Ceux-ci sont émis sur le LAN par chaque routeur et renferme une métrique propre au flux. La valeur de cette métrique varie en fonction d'une valeur de préférence définie pour chacun des routeurs et de la distance séparant le routeur de la source du flux. Pour chaque groupe actif, le routeur possédant la meilleure métrique sera élu.

3.3.2. La retransmission des flux multicast

Le rôle de chaque routeur étant défini, ceux-ci peuvent commencer à gérer le trafic multicast. Pour ce faire, la principale tâche d'un routeur est la gestion de la retransmission des paquets multicasts. Afin de réaliser ce travail, un routeur multicast se base sur le contenu de sa TIB et de sa MRIB.

Lorsqu'un paquet multicast parvient au routeur, celui-ci examine sont contenu pour en connaître sa source et le groupe multicast auquel il est destiné. A l'aide des informations contenue dans la TIB, il lui est possible de déterminer s'il doit gérer ce flux. Pour ce faire, il commence par regarder si le flux (S,G) est actif, c'est-à-dire s'il doit être géré via le SP-tree. Dans l'affirmative, et si la MRIB confirme que ce paquet est bien arrivé via l'interface menant au routeur PIM SM voisin en amont sur cet arbre, le routeur retransmet le paquet sur toutes les interfaces par lesquelles un message JOIN (S,G) a été réceptionné.

Par contre, si ce flux ne doit pas être géré via le SP-tree, le routeur vérifie si le flux (*,G) est actif et si le paquet est bien arrivé par l'interface menant en amont au RP-tree. Si c'est le cas, celui-ci retransmet également le paquet via les interfaces de sortie actives pour ce flux (*,G) reprise dans la TIB

Si ce flux n'est à retransmettre ni via le SP-tree, ni via le RP-tree, le routeur ne doit pas gérer ce flux. Ce peut être le cas lorsque le paquet est arrivé par une interface directement connectée à un LAN. Le flux peut être transmis par l'intermédiaire d'un routeur à l'intention d'un troisième ou d'une station ayant demandé la réception de ce flux via un message 'IGMP'.

3.3.3. La demande de réception d'un flux

Un routeur PIM SM gère la retransmission d'un flux multicast à l'aide des informations contenues dans sa TIB. Initialement, celle-ci est vide. Les routeurs doivent la compléter dynamiquement en fonction des désirs des receveurs multicast du domaine.

Pour ce faire, lorsqu'une station connectée à un LAN du domaine désire recevoir le trafic d'un groupe multicast, elle le signale aux routeurs connectés à ce LAN à l'aide des services du protocole IGMP. Lorsque le DR de ce LAN reçoit un QUERY IGMP en provenance du LAN pour un nouveau groupe G, il entame le processus lui permettant d'acheminer le trafic de ce groupe au LAN demandeur.

Pour ce, Le DR recherche dans sa TIB l'identification (*,G) de ce groupe. Si cette identification est active, cela signifie que le routeur gère déjà le trafic de ce groupe. Il

lui suffit donc d'ajouter l'interface par laquelle le QUERY est entré à la liste des interfaces des LAN demandeurs de ce flux gérés par ce DR.

Dans le cas contraire, le routeur commence par ajouter une nouvelle ligne à la table avec l'identifiant (*,G) référençant le nouveau groupe et y ajoute l'interface menant au LAN demandeur. Il doit ensuite prévenir les routeurs en amont du RP-tree qu'il désire recevoir ce flux. Pour ce, le DR envoie un message JOIN (*,G) en direction du RP du groupe (Connu grâce aux service du BSR) via l'interface menant au routeur voisin en amont de l'arbre (obtenu grâce à la MRIB).

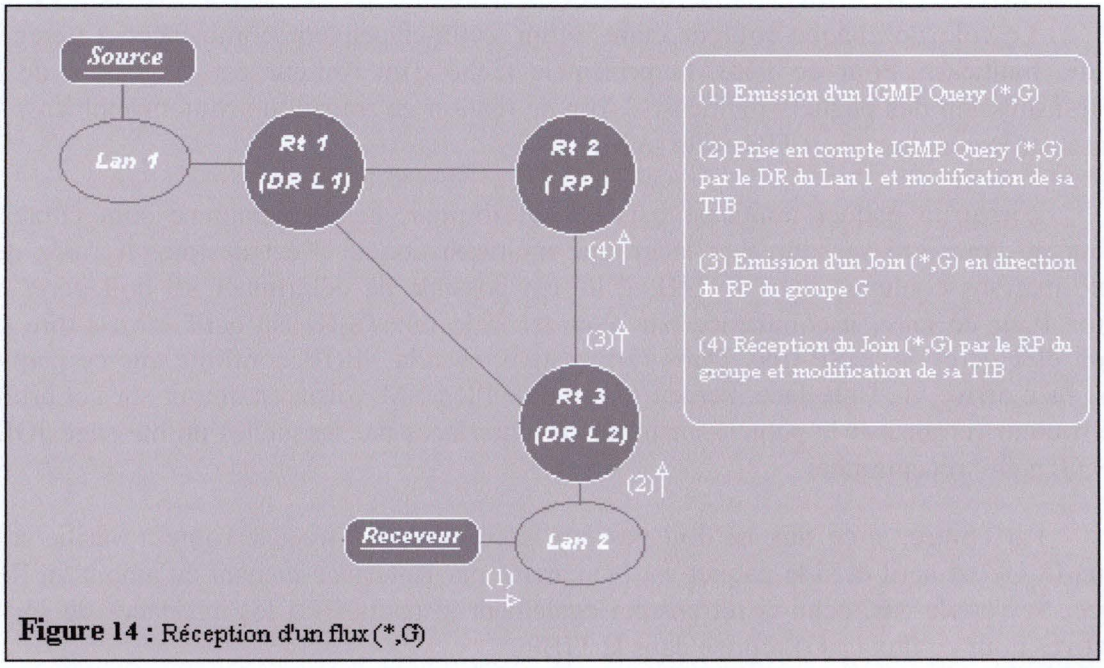


Figure 14 : Réception d'un flux (*,G)

A la réception du JOIN, chacun des routeurs intermédiaires (y compris le RP) effectue la même opération. C'est à dire qu'il consulte sa TIB pour y trouver la ligne identifiant du flux (*,G). Si celle-ci n'existe pas, la crée et retransmet le JOIN au routeur suivant. Il ajoute ensuite l'interface d'entrée du message JOIN à la liste des interfaces de sortie du flux (*,G) nouvellement créé et transfère le message au routeur précédant. A l'inverse, s'il trouve l'identifiant du flux dans sa TIB, il est juste nécessaire d'ajouter l'interface à la liste des interfaces de sortie du flux (*,G). Le JOIN est transmis de la sorte de routeur en routeur et ce, jusqu'au RP ou jusqu'au premier routeur recevant déjà le flux multicast demandé.

Depuis IGMP version 3, un receveur a également la possibilité de demander directement la réception d'un flux (S,G). Dans ce cas, le DR réceptionne le QUERY (S,G) et effectue la même opération que ci-dessus, mais, cette fois, en se basant sur les identifiants (S,G) de la TIB. Ainsi que sur des messages JOIN (S,G) ayant la source S comme racine.

3.3.4. L'émission d'un nouveau flux (S,G)

Contrairement au receveur, une station du domaine désirant émettre du trafic en direction d'un groupe multicast ne doit pas passer par un protocole du type IGMP. Il lui suffit de commencer à émettre ses paquets en direction du groupe (Pour rappel, une source n'est pas obligatoirement un membre de ce groupe). C'est en fait le DR du LAN auquel la source est connectée qui est responsable d'effectuer les opérations nécessaires.

A la réception d'un paquet multicast arrivant par une interface directement connectée à un LAN, le DR vérifie si ce paquet a été émis par l'une des stations de ce LAN. Dans l'affirmative, il parcourt sa TIB à la recherche d'une identification (S,G) correspondante. S'il ne trouve pas cette identification, cela signifie qu'il s'agit du premier paquet de ce flux. Dans ce cas, le routeur ajoute une nouvelle entrée (S,G) à sa TIB identifiant ce nouveau flux.

A ce moment, un problème se pose. En effet, un routeur PIM SM retransmet un paquet multicast uniquement via les interfaces par lesquelles un message JOIN (*,G) ou (S,G) a été réceptionné. Or, personne n'est au courrant de l'existence de ce nouveau flux (S,G). Il n'est donc pas possible dans cet état d'émettre le trafic en mode multicast. Afin de faire connaître ce flux au domaine, le routeur le place en mode 'Registering' en mettant à ON le flag Register du flux dans sa TIB. Il encapsule ensuite le paquet multicast dans un message 'Register' qu'il adresse en mode unicast au RP du groupe. Il fait de même avec tous les paquets de ce flux qui suivent et ce, durant toute la durée du mode 'Registering'. Celle-ci ne se terminant qu'à la réception d'un message 'Register Stop' (S,G) en provenance du RP du groupe.

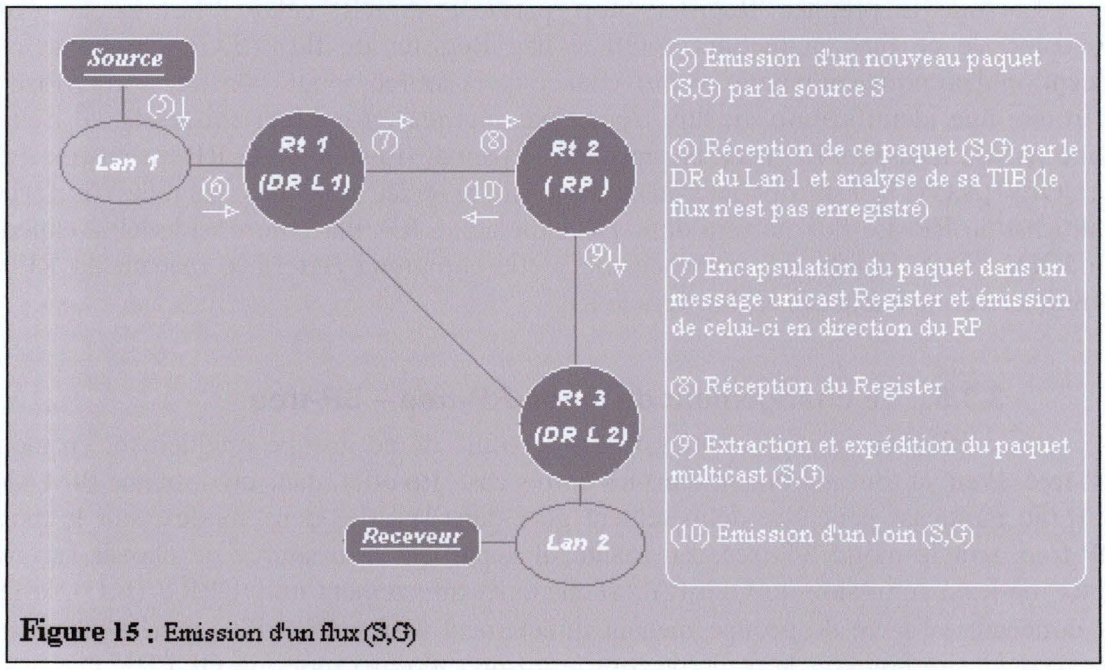


Figure 15 : Emission d'un flux (S,G)

Les messages 'Register' sont donc transmis par le protocole unicast du DR au RP du groupe. Lorsque le RP réceptionne l'un de ceux-ci, deux situations sont envisageables. La première se présente lorsque au moins un receveur a déjà fait savoir qu'il désirait recevoir le trafic multicast de ce groupe. C'est à dire lorsqu'un message

JOIN (*,G) lui est parvenu. Dans ce cas, une identification (*,G) ainsi que la liste des interfaces par lesquelles un JOIN est arrivé est disponible dans la TIB du RP. Le RP désencapsule alors le paquet multicast repris dans le message 'Register' et le réexpédie en multicast sur toutes les interfaces reprisent dans la ligne (*,G) de sa TIB. Il initie également un JOIN (S,G) afin de relier la source à l'arbre multicast de ce groupe. Le RP continuera à travailler de la sorte jusqu'à la réception du premier message multicast (S,G). A ce stade, il n'est plus nécessaire de travailler en mode 'Registering'. Le RP expédie donc un message 'Register Stop' (S,G) au DR de la source. Celui-ci est également expédier en mode unicast.

Lorsque le DR réceptionne le JOIN (S,G) en provenance du RP, celui-ci ajoute simplement l'interface d'entrée de ce message à la liste d'interface du flux dans sa TIB. A ce stade, le flux est toujours en mode 'Registering'. Cela signifie qu'à l'arrivée d'un nouveau paquet en provenance de la source, le LHR l'expédie en mode multicast via l'interface d'entrée du JOIN et continue à l'encapsuler dans un message 'Register'. Il continuera à dupliquer les paquets (S,G) de la sorte jusqu'à la réception du premier message 'Register Stop'.

Contrairement à la première situation, la deuxième se présente lorsque aucun JOIN (*,G) n'est parvenu au RP à l'arrivée d'un message 'Register' pour ce groupe. Dans ce cas, soit la TIB du RP ne contient pas d'identification (*,G) pour ce groupe ou soit elle en contient une avec une liste des interfaces de sorties vide. Dans cette situation, le RP expédie directement un message 'Register Stop' (S,G) au DR de la source et rejette le paquet.

Lorsque le premier 'Register Stop' parvient au DR, celui-ci termine la phase 'Register' de ce flux en mettant à Off le flag Register du flux (S,G). Dés lors, à la réception d'un nouveau paquet en provenance de la source, le DR consulte son OIList. Il y trouve une identification du flux n'étant pas marqué en mode 'Registering'. Il n'émet donc pas de message 'Register' encapsulant le paquet. Ensuite, si le RP n'a pas expédié de JOIN (S,G), la liste des interfaces de sortie associée à ce flux ne relie pas l'arbre multicast au RP. Le flux ne sera donc pas expédié au RP. Par contre, si le RP a expédié un JOIN (S,G), la liste des interfaces de sortie comprend l'interface menant au RP. Il continuera donc à émettre le flux vers le RP.

3.3.5. Le changement de mode RP-tree – SP-tree

Si le RP d'un groupe multicast gère le trafic de ce groupe uniquement en mode SP-tree, Il en va tout autrement dans les autres cas. En effet, dans un domaine PIM SM, le trafic multicast peut être délibérément géré dans l'un des deux modes, soit le mode RP-tree, soit le mode ST-tree. En réalité, il appartient à la source de choisir l'un des deux modes à l'émission du QUERY. Si elle émet directement un QUERY (S,G), le DR se connecte à l'arbre du groupe menant directement à la source. Plus généralement, un receveur désire recevoir le trafic de tout le groupe. Il émet alors un QUERY (*,G). Le DR se connecte donc à l'arbre partagé du groupe G.

Lorsque le DR doit rejoindre un nouveau groupe à la réception d'une demande issue d'un LAN directement connecté, il émet un JOIN (*,G) en direction du RP du

groupe. Le flux est à ce moment géré en mode RP-tree. Celui-ci n'a alors aucune idée des sources actives sur ce groupe. Il fera la découverte de chacune de ses sources au fur et à mesure que les paquets multicast émis par ces sources lui parviendront via le RP-tree. Le DR a maintenant la possibilité de basculer vers le mode SP-tree. Voyons de plus prés comment cela ce passe.

En réalité, le DR décide de switcher la gestion d'un flux d'un groupe multicast entre les deux modes en fonction du débit de ce flux. Dans un DR, chaque groupe multicast dispose d'un seuil exprimé en Kbps au-dessus du quel le routeur passe en mode ST-tree. Ce seuil peut être configuré afin de contrôler la gestion du flux. Par défaut, celui-ci est généralement fixé à zéro. Mais il est possible de définir un seuil supérieur ou infini.

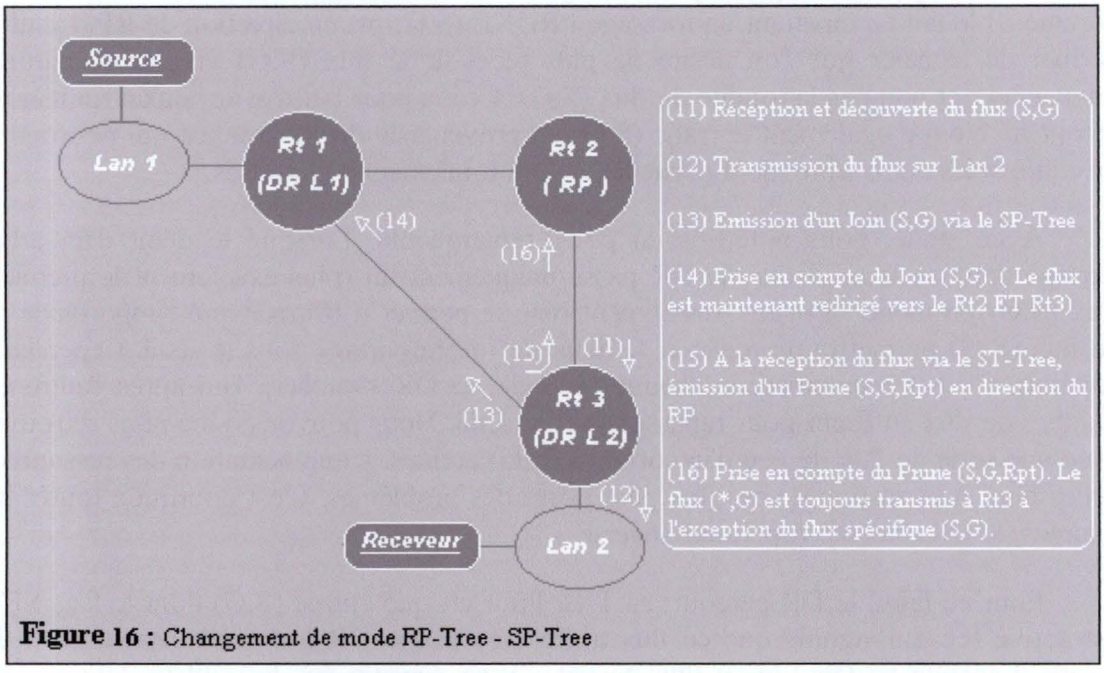


Figure 16 : Changement de mode RP-Tree - SP-Tree

Lorsqu'un seuil est fixé à zéro, cela signifie que le DR passera directement en mode ST-tree dés que celui-ci recevra le premier paquet multicast en provenance du RP-tree et ce, pour toutes les sources. S'il est fixé à une valeur comprise entre zéro et l'infini, le DR mesure continuellement le débit du flux (*,G) en provenance du RP. Le débit est donc calculé en fonction du RP-tree du groupe et non du trafic total de ce groupe. Lorsque celui-ci dépasse le seuil fixé, le DR passe en mode SP-tree pour ce groupe.

Par contre, lorsque le seuil est fixé à l'infini, ce flux restera toujours géré en mode RP-tree. Il peut paraître étrange et contraire à l'idée du protocole d'empêcher un flux de passer en mode SP-tree. Cependant, dans certaines situations ou le nombre de sources est très élevé, cette solution peut être préférable car elle permet de limiter les ressources nécessaires aux routeurs pour la gestion d'une multitudes de flux (S,G). C'est d'autant plus intéressant lorsque ces sources n'émettent que peux de trafic. Car elles n'engorgent pas le RP-tree.

Voyons de plus près le mécanisme mis en œuvre par le routeur pour effectuer ce changement de mode. Toute les secondes, le DR calcul le débit de chaque flux (*,G) destiné à l'un de LAN directement connecté à celui-ci. Si le trafic de l'un de ses groupes dépasse le seuil fixé, le routeur place se flux (*,G) en mode 'SP-tree SwitchOver' en mettant à ON le flag 'SPT' dans la ligne (*,G) de sa TIB.

Si ce flag est activé à l'arrivée d'un paquet (S,G) via l'arbre partagé, le DR crée une entrée (S,G) dans sa TIB et active le flag 'SPT' de ce flux (S,G) afin de retenir que cette entrée a été créée suite au dépassement du seuil fixé pour ce groupe. Il désactive le flag 'STP' de l'entrée (*,G) (ceci permet au processus de recommencer la seconde suivante et de ne pas switcher automatiquement tout les flux (S,G) de ce groupe). Il émet ensuite un message JOIN (S,G) pour relier cette source via le SP-tree. Il reste au DR à signaler qu'il ne désire plus recevoir le flux (S,G) en provenance de l'arbre partagé. Il le fait en émettant un message PRUNE (S,G,rpt) en direction du RP. Celui-ci permet de signaler que l'on désire ne plus recevoir le flux (S,G) via l'arbre partagé uniquement et non pas la totalité du flux (S,G). Ceci a pour but d'éviter qu'un routeur en amont ne bloque également le trafic (S,G) en provenance du SP-tree, ce qui pourrait se produire si les deux arbres sont partiellement ou totalement superposés.

A ce stade, nous noterons un point remarquable. Lorsque le débit d'un arbre partagé dépasse le seuil fixé, le DR passe uniquement un (plus exactement le premier) flux (S,G) en mode SP-tree. Absolument rien ne permet d'affirmer que l'importance de ce flux (S,G) permettra de repasser le débit de l'arbre partagé sous le seuil. Cependant, l'opération étant recommencée chaque seconde, le DR switchera l'un après l'autre un nombre de flux suffisant pour repasser sous le seuil. Nous pouvons donc nous retrouver avec une série de flux de peu d'importance (S,G) activés. L'augmentation des ressources nécessaires à la gestion de ces flux peut poser des problèmes. C'est pourquoi, toutes les minutes, le DR effectue l'opération inverse.

Pour ce faire, le DR parcourt sa TIB. Pour chaque entrée (S,G) dont le flag 'SPT' est activé (ce qui signifie que ce flux a été crée par le DR lors d'un dépassement du seuil), il calcule le débit de ce flux. Si celui-ci ne dépasse pas le seuil fixé du groupe auquel il appartient, il replace ce flux (S,G) en mode RP-tree. Pour ce faire, il doit émettre un Prune (S,G) en direction de la source et désactiver l'entrée (S,G) devenue inutile de sa TIB. De plus, il est également nécessaire de rattacher chacun de ses flux à l'arbre partagé de leur groupe respectif. Ceci est réalisé par le routeur en renvoyant pour chaque groupe possédant au moins un flux réactivé un message JOIN (*,G) accompagné des PRUNE (S,G,rpt) uniquement pour les flux (S,G) de ce groupe restant en mode SP-tree.

3.3.6. la demande de fin de réception d'un flux

A ce stade, les stations du domaine peuvent demander la réception d'un flux (*,G) ou du flux (S,G) d'une source spécifique, Les routeurs sont capables de gérer les arbres de ces différents flux et les DR sont également capables de gérer les changements de mode (mode RP-tree, mode SP-tree) des flux (*,G) dont ils ont la charge. Une dernière tâche reste encore à être prise en charge. C'est la clôture d'un abonnement à l'un de ces flux.

Lorsqu'un flux multicast n'est plus désiré par les stations d'un LAN, le DR en est averti par l'intermédiaire des messages IGMP (ou, plus exactement, par l'absence de ceux-ci). Dans ce cas de figure, le DR modifie sa TIB en y supprimant l'interface le reliant à ce LAN de la liste des LAN demandeurs. S'il reste des interfaces actives dans l'une des deux listes de sorties (en direction des LAN demandeurs et en direction des routeurs voisins), le routeur continue à gérer ce flux. Dans le cas contraire, il n'est plus nécessaire que le flux continue à parvenir à ce routeur.

A ce moment, le DR expédie un message PRUNE vers la racine de l'arbre. Dans le cas d'un flux (*,G), il est donc expédié vers le RP du groupe. Par contre, s'il s'agit d'un flux (S,G), il est transmis vers la source S. A la réception de celui-ci, le routeur voisin supprime de sa TIB l'interface d'entrée du message PRUNE de liste des interfaces de sorties de ce flux. S'il ne reste plus d'interfaces par lesquelles le routeur doit transmettre le flux, il émet à son tour un message PRUNE pour ce flux en direction de la racine. Tout comme les JOIN, le message PRUNE est transmis de routeur en routeur voisin et ce, jusqu'au premier routeur devant continuer à transmettre ce flux sur au moins une interface ou jusqu'à la racine du flux.

Un point important est à remarquer. Si l'interface d'entrée d'un message PRUNE est raccordée à une ligne 'point-à-point', le routeur peut directement supprimer celle-ci de la liste des interfaces de sorties. Par contre, s'il s'agit d'une interface menant à un LAN, il peut y avoir d'autres routeurs également connectés à ce LAN et désirant continuer à recevoir ce flux. Dans ce cas, il ne peut donc pas simplement supprimer l'interface de sa liste de sortie. Il est au contraire nécessaire de passer par un mécanisme plus complexe. Dans ce cas, le routeur démarre un timer et ne supprime l'interface qu'à l'expiration de celui-ci. Pendant ce temps, un autre routeur désirant continuer à recevoir ce flux à la possibilité de contrer le mécanisme et ce, simplement par l'émission d'un nouveau JOIN sur ce LAN.

3.3.7. Les changements de topologie du domaine

Les différentes phases du protocole vues jusqu'à présent permettent la découverte de la topologie du domaine, la répartition des rôles entre les différents routeurs ainsi que la gestion des sources, des receveurs et des arbres de diffusions. Il est cependant important de remarquer qu'un domaine Internet est un environnement dynamique. Des connexions peuvent être rompues ou crées, des routeurs peuvent tomber en panne, de nouveaux routeurs peuvent être initialisés, ...

Nous l'avons déjà vu, PIM SM (comme tous les protocoles réseau efficaces) doit être résistant aux changements de topologie du réseau. Il est donc nécessaire de lui fournir les moyens de s'adapter dynamiquement à ces changements. Ceci est réalisé grâce à l'adjonction d'une série de timers aux différents éléments du protocole.

Tous les timers utilisés dans PIM SM fonctionnent comme des comptes à rebours. Ils sont initialisés à une certaine valeur et diminué progressivement. Lorsque leur valeur tombe à zéro, il provoque l'exécution d'une action. C'est ainsi que l'on retrouve dans chaque routeur PIM un timer associé à l'émission des messages 'Hello'. A l'émission de se message, le routeur l'initialise à une valeur généralement égale à 30 secondes.

Lorsque ce temps est écoulé, le routeur recommence l'opération. Si un changement de topologie se présente, les routeurs du domaine mettrons donc au maximum 30 secondes avant de s'en rendre compte et de pouvoir réagir en fonction. Par exemple, si le DR d'un LAN est déconnecté de ce LAN, les autres routeurs directement connecté à ce LAN rééliront rapidement un nouveau DR. Il en va de même pour la gestion des messages émis par les CBSR et les Candidats RP. Ainsi que pour l'émission des messages émis par le BSR actif en direction des DR du domaine. Ces différents timers permettent de définir dynamiquement le rôle des différents éléments du domaine en s'adaptant au changement de topologie de celui-ci.

Il est également nécessaire de gérer dynamiquement la structure de tous les arbres actifs du domaine. Une série de timers permettant de réaliser cette gestion dynamique est associée aux composants de la TIB de chaque routeur. C'est ainsi que l'on y retrouve entre autre un timer permettant la retransmission périodique des messages JOIN pour chaque flux actif d'un réseau et un autre permettant de supprimer la retransmission d'un flux par une interface active n'ayant réceptionné aucun JOIN pour ce flux depuis un certain temps.

La liste des différents timers décrits ci-dessus n'est pas complète. Une liste plus exhaustive est fournie en annexe de ce travail. La gestion de cet ensemble de timers permet non seulement au protocole d'assurer son adaptation à la topologie dynamique du réseau, mais en plus, d'optimiser celui-ci. Il existe par exemple un timer permettant à un routeur directement connecté à un LAN de temporiser la prise en compte d'un message "Prune" afin de laisser le temps à un autre routeur directement connecté à ce LAN de contrer le message "Prune".

4. Analyse fonctionnelle du simulateur

L'objectif de cette partie du travail (ainsi que de la suivante) consiste à réaliser un simulateur de PIM SM. Ce simulateur n'a pas pour but de servir à des professionnels des télécommunications afin de tester l'une ou l'autre configuration de leur réseau. Celui-ci a un but purement didactique. Il doit faciliter l'apprentissage du fonctionnement de ce protocole complexe.

Le simulateur doit permettre de représenter les différents éléments d'un domaine multicast et exécuter sur cette représentation une simulation des différentes phases d'exécution de PIM SM. Cette simulation doit être aussi proche que possible d'une implémentation réelle du protocole. Cependant, elle devra rester suffisamment simple pour garder ses qualités didactiques.

Outre les fonctionnalités techniques (entendons par là les besoins techniques de PIM SM), il est intéressant de se pencher de près sur le ou les type(s) d'utilisateur(s), sur leurs besoins ainsi que sur leur environnement de travail et ce, afin de réaliser un simulateur utile, utilisable et donc utilisé. L'objectif poursuivi ici n'est pas de réaliser une étude complète et détaillée des utilisateurs et du contexte de travail ce qui dépasserait largement les limites de ce travail. Cependant, une petite réflexion sur le sujet permettra la réalisation d'une interface bien conçue

4.1. Stéréotypes des utilisateurs

Les utilisateurs de ce système sont soit enseignants soit apprenants. Un enseignant est, dans ce cas de figure, un utilisateur exposant le fonctionnement du protocole PIM SM à un auditoire et se servant du système afin d'illustrer ses propos étape par étape. L'enseignant peut également désirer se servir du système afin de soumettre une série de cas précis à ses étudiants et ce, afin que ceux-ci les analysent individuellement ou les complètent à titre d'exercice.

Un apprenant peut être une personne assistant à l'exposé d'un enseignant. Dans ce cas, l'apprenant n'interagit pas directement avec le système (c'est ici le rôle de l'enseignant). Cependant, il peut également se servir du système pour étudier personnellement un cas proposé par un enseignant ou étudier un cas qu'il crée par ses propres soins.

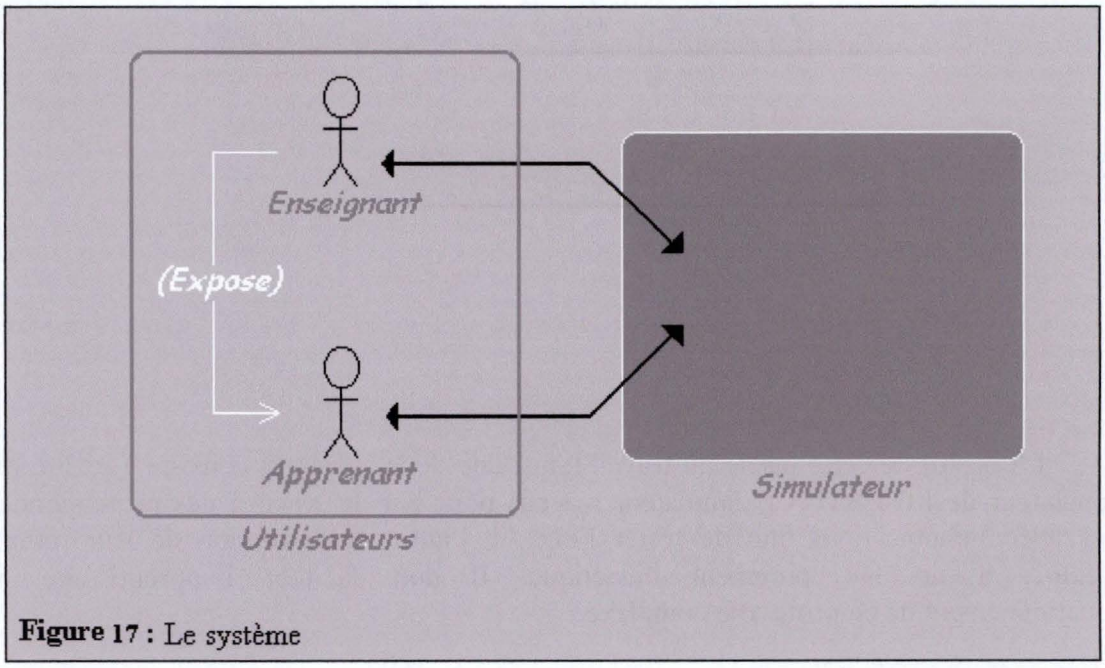


Figure 17 : Le système

D'un point de vue fonctionnel, les utilisateurs interagissent avec le simulateur de la même manière et ce, qu'il s'agisse d'un enseignant ou d'un apprenant. Il conviendra simplement de veiller à respecter leurs besoins respectifs.

D'un point de vue cognitif, les utilisateurs de ce système sont des personnes ayant une bonne connaissance du domaine des réseaux. Un enseignant peut sans crainte être considéré non seulement comme spécialiste réseau, mais également du protocole PIM SM. L'apprenant est dans la majorité des cas une personne ayant de bonnes connaissances des différents composants d'un réseau et du fonctionnement de celui-ci. Cependant, nous ne pouvons pas considérer que ceux-ci possèdent la moindre connaissance du multicasting et plus particulièrement du protocole PIM SM.

Nous pouvons également estimer que l'utilisation d'une interface informatique n'est pas un problème pour les utilisateurs du simulateur. Ceux-ci sont habitués à utiliser l'outil informatique dans leur vie professionnelle ou d'étudiant. L'utilisation

d'une interface graphique quelque peu complexe ne sera donc pas un problème insurmontable pour ceux-ci.

4.2. Le contexte d'exécution de la tâche

Le contexte d'exécution dans lequel le simulateur sera employé est très variable. En effet, celui-ci peut être l'environnement techniquement dégradé d'un auditoire dans lequel un enseignant réalise son exposé. Il peut également s'agir de l'environnement de travail très confortable d'un bureau dans lequel l'enseignant prépare ses exposés ou celui de l'étudiant à son domicile privé.

Lors de la conception de l'interface, il conviendra de d'assurer que celle-ci sera utilisable dans chacun de ses environnements. C'est-à-dire qu'elle doit répondre au besoin de l'environnement le plus dégradé qui, dans ce cas, est celui dans lequel l'enseignant utilisant le simulateur lors d'un exposé. Celui-ci travaillera le plus souvent debout, face à son auditoire, avec son matériel posé sur un bureau n'étant pas toujours approprié. De plus, il doit partager son attention entre la poursuite de son exposé (et les réactions de son auditoire) et la manipulation du simulateur. Il est donc important de veiller à ce que l'interface soit la plus visuelle possible et que celle-ci dispose de moyens d'interactions simples, rapides et efficaces.

4.3. Les besoins des utilisateurs

Outre les besoins définissant le type d'interface à développer que nous venons d'aborder ci-dessus, Il est important de définir quelles sont les actions qu'un utilisateur doit pouvoir accomplir à travers l'utilisation de cette interface. Un utilisateur doit pouvoir :

- Charger dans le simulateur un domaine à partir d'un fichier existant
- Modifier un domaine chargé dans le simulateur
- Créer un nouveau domaine
- Enregistrer un domaine dans un fichier (existant ou nouveau)
- Activer l'exécution de la simulation sur le domaine chargé dans le simulateur
- Réinitialiser la simulation en cours d'exécution
- Observer le déroulement du protocole sur les éléments du domaine

Au cours de la création ou de la modification d'un domaine, l'utilisateur doit avoir la possibilité de :

- Ajouter un élément au domaine
- Définir les attributs d'un élément du domaine
- Modifier les attributs d'un élément du domaine
- Supprimer un élément du domaine
- Connecter deux éléments du domaine

Par observer le déroulement du protocole, nous entendons :

- Observer visuellement le déroulement de celui-ci sur une représentation graphique du domaine
- Interroger chaque élément du domaine pour en obtenir sa configuration détaillée

- Pouvoir faire progresser l'exécution de la simulation pas à pas

Les tâches et sous-tâches du protocole à simuler lors de l'exécution sont les suivantes:
- l'élection du BSR pour le domaine,
- l'élection des DR sur chaque LAN,
- la gestion de l'élection du RP pour chaque groupe actif sur le domaine,
- la gestion du protocole IGMP sur les différents LAN,
- les procédures Join et Prune diffusés entre les routeurs du domaine,
- le changement de mode SP-tree et RP-tree,
- la gestion de nouvelles sources par les DR,
- la gestion des receveurs d'un flux,
- la gestion d'un groupe par son RP,
- la gestion des principaux éléments de la TIB,

De plus, l'utilisateur n'ayant pas forcément une grande connaissance du domaine multicast, il est important d'ajouter au simulateur une gestion de l'aide nécessaire. Cette aide doit fournir un certain nombre d'explications concernant le fonctionnement du protocole, le rôle de chaque composant et également sur le fonctionnement du simulateur. De plus, l'utilisateur créant un domaine doit recevoir des renseignements sur l'état et les manquements de celui-ci. Par exemple, un domaine ne peut fonctionner sans CBSR ou sans CRP. Si l'utilisateur ne les configure pas, le simulateur doit le renseigner sur les manquements lors de l'activation de la simulation.

4.4. Ce qu'il n'est pas nécessaire de simuler

La gestion des différents timers que l'on retrouve dans PIM SM ne seront pas implémenté dans le simulateur. En effet, outre la complexité qu'ils entraînent, ceux-ci ne sont pas primordiaux pour la compréhension du fonctionnement du protocole. Au contraire, ceux-ci pourraient augmenter la complexité du simulateur et nuire à la compréhension de l'apprenant.

D'une manière générale, les timers sont là pour parer aux divers problèmes que l'on rencontre sur un réseau. En effet, dans une implémentation fonctionnelle du protocole, un routeur ne connaît pas automatiquement l'existence des autres routeurs et ne sont pas automatiquement mis au courrant des modifications de la configuration de ces routeurs et des connexions existantes. Pour ce faire, ils sont obligés de s'échanger un grand nombre de messages et de toujours prévoir ne pas être averti d'un changement. Dans le simulateur, nous pouvons aisément parer ce problème.

5. Analyse Conceptuelle

Dans cette partie du travail, nous allons définir les différents objets nécessaires à la réalisation du simulateur. Nous définirons leurs rôles ainsi que les interactions existant entre ceux-ci. Dans le chapitre suivant, nous étudierons plus en détail la structure de chacun de ses objets, des relations qui les unissent et de leur mode de fonctionnement. Mais, pour l'instant commençons par séparer cette tâche en deux.

Nous pouvons facilement constater qu'il nous est nécessaire de définir l'ensemble des objets propres au domaine. C'est-à-dire les éléments physiques constitutifs d'un réseau et les différents objets nécessaires au protocole PIM SM. C'est la première tâche que nous devrons réaliser. Ceci fait, notre deuxième tâche consistera à définir les objets propres à l'interface permettant à l'utilisateur d'interagir avec les différents éléments du domaine.

5.1. Les objets du domaine

5.1.1. Le hardware

Le principal élément est le domaine. Celui-ci contient un ensemble de connexion ainsi qu'un ensemble d'éléments physiques (cf. figure 18). Un certain nombre d'interfaces sont incluses dans chacun des éléments physiques. Ces interfaces permettent de relier l'élément à une connexion réseau responsable de l'acheminement des différents messages.

Pour les besoins du simulateur, nous pouvons limiter les types de connexion aux lignes Point-à-Point et aux LAN représentants un réseau local et n'étant pas limités à 2 interfaces.

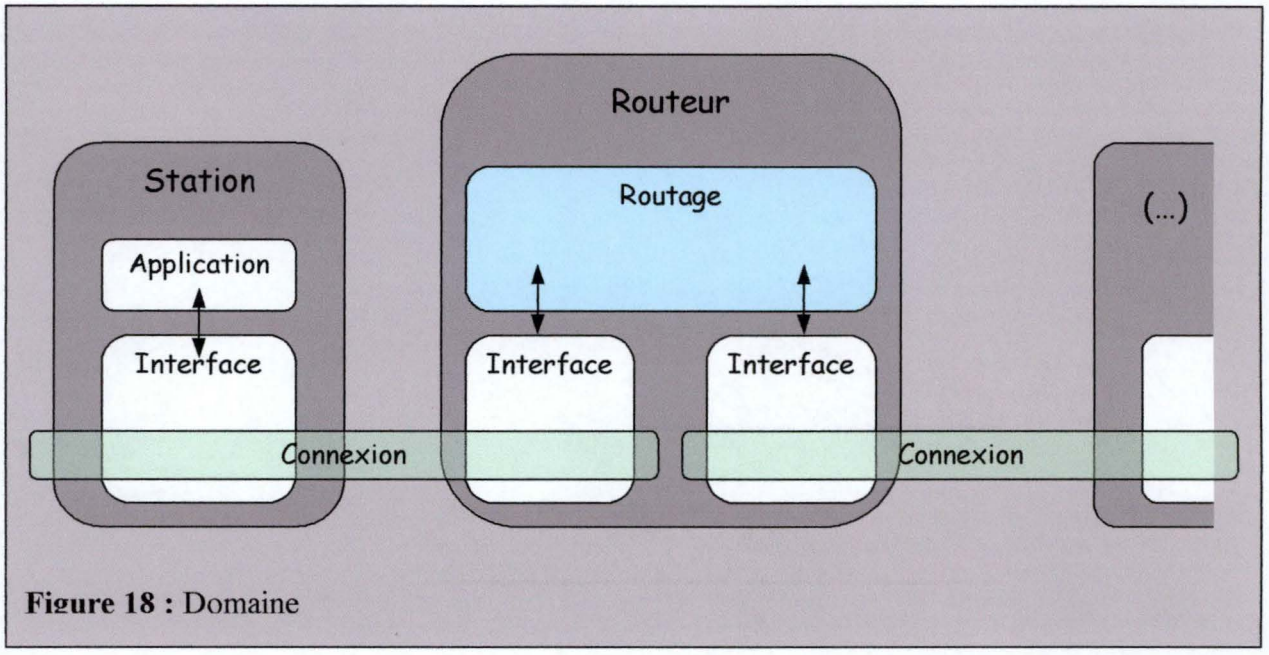


Figure 18 : Domaine

Le simulateur ne nécessite que l'utilisation de deux types d'éléments bien précis. Il s'agit des stations (Celles-ci oeuvreront comme receveur ou comme émetteur de flux) que nous limiterons à une seule interface et, bien évidemment, des routeurs qui eux possèdent plusieurs interfaces.

Une interface est responsable de la gestion des transmissions de messages entre l'élément du réseau auquel elle appartient et la connexion auquel elle est reliée. Chaque interface possède sa propre adresse IP. Celle-ci est composée de l'adresse de la connexion auquel l'interface est reliée et de sa propre adresse. Celle-ci étant unique pour la connexion. Par transmissions de messages, il faut comprendre réception des messages en provenance de la connexion et retransmission à l'élément (méthode « in »). Mais également l'inverse (méthode « out »). C'est-à-dire la transmission vers la connexion des messages émis par l'élément.

Revenons quelques instants aux connexions. Leur rôle principal est la diffusion des messages (méthode « diffuse ») entre les interfaces lui étant directement reliées. Lorsque l'une d'entre elles émet un message, la connexion doit le retransmettre aux autres interfaces. Pour ce faire, il est nécessaire de pouvoir connecter (méthode « connecte ») et déconnecter (méthode « deconnecte ») une interface existante à cette connexion. Par simplicité, cette méthode sera également responsable de l'attribution d'une adresse IP par défaut à l'interface que l'on relie.

PIM SM nécessite l'utilisation du protocole IGMP afin de déterminer les groupes multicast réclamés par les stations du LAN. Ce protocole IGMP est relativement complexe et sort du cadre de ce travail. C'est pourquoi nous chargerons l'objet de connexion Lan (méthode « igmp ») de simuler cette tâche. Il en est de même pour la gestion des messages Assert pour un flux multicast.

Il en est de même pour la gestion de l'élection du BSR pour le domaine, celle des DR pour chaque LAN ainsi que la diffusion de la liste des RP. Ces tâches sont

également relativement complexes et consommatrices de ressources. C'est pour cette raison (et parce que l'implémentation de ceux-ci n'apporterait pas grand chose de plus au simulateur) que ces tâches seront simulées directement dans l'objet « Domaine » (méthodes « setBsr », « getBsr », « setRp », « getRp »). Remarquez que les méthodes « setBsr » et « setRp » effectuent directement la sélection en se basant sur les connaissances des éléments physiques du domaine tandis que la méthode « setDr » est propre à chaque objet connexion Lan. C'est cette méthode qui effectuera le travail et définira la valeur de la variable booléenne isDr de chacune des interfaces lui étant reliées. Elle doit donc être effectuée sur chaque LAN.

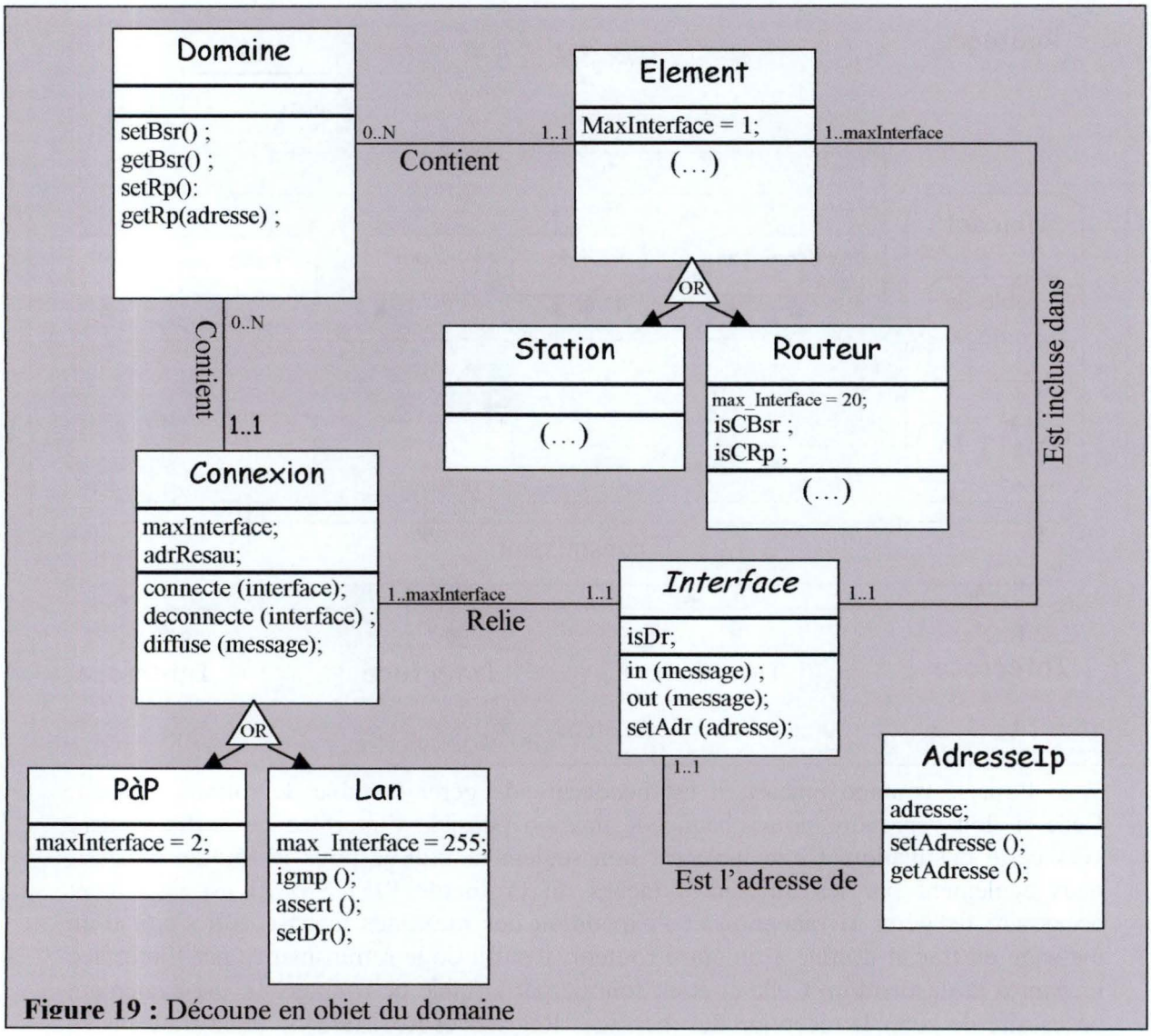


Figure 19 : Découpe en objet du domaine

5.1.2. Détails des éléments physiques

Nous avons abordé l'étude des éléments physiques du domaine. Cependant, ceux-ci étant en réalité très complexes, il est nécessaire de se pencher de plus près sur leur définition. Il est constatable que s'il est vrai que ces éléments du domaine (station et routeurs) doivent tous deux gérer les interfaces leur étant connectées et plus particulièrement la réception et la transmission de message, la similitude s'arrête là.

Les tâches à accomplir par les stations sont relativement simples et peu nombreuses pour les besoins du simulateur. Outre l'émission et la réception des messages, il lui suffit de gérer les groupes à recevoir ainsi que les groupes pour lesquels elle œuvre en temps que source. Il en va cependant tout autrement pour les routeurs. Non seulement ceux-ci doivent gérer plus d'une interface (et donc gérer l'émission d'un message par une interface précise), ceux ci doivent gérer le routage unicast et multicast (cf. figure 20). Ce qui n'est pas une mince affaire et complexifie grandement la tâche de la couche « Transmission » et des fonctions « In » et « Out ».

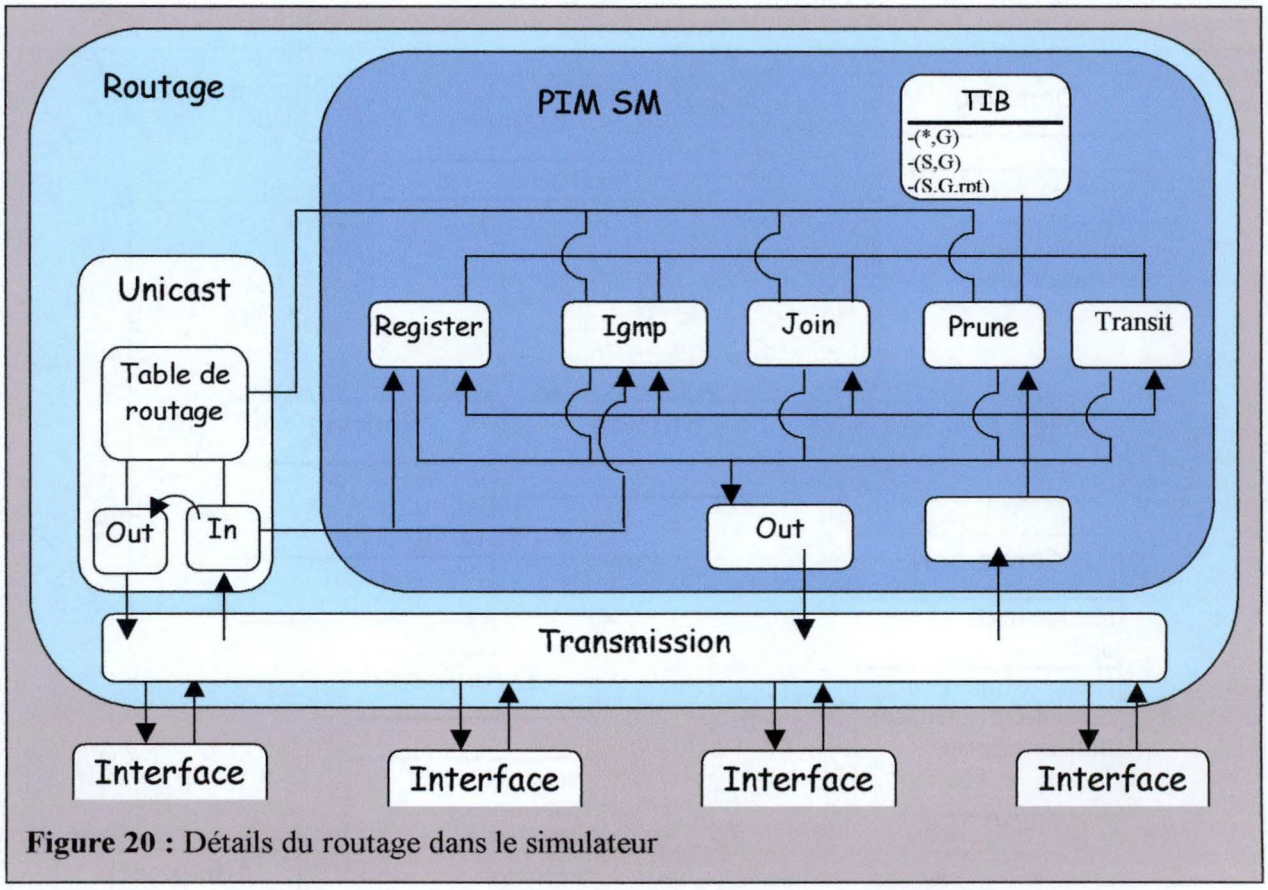


Figure 20 : Détails du routage dans le simulateur

Pour le routage unicast, il est nécessaire de gérer la table de routage unicast. Celle-ci doit reprendre pour chaque destination possible l'interface de sortie menant vers cette destination. Cette table est non seulement utilisée pour le routage unicast, mais également par les différentes tâches du protocole PIM SM. Il est également nécessaire de gérer la réception et l'expédition des messages unicast. S'il s'agit d'un message en transit destiné à un autre routeur, il suffit de le retransmettre par l'interface menant à la destination. Celle-ci étant fournie par la table de routage. Il est également nécessaire de gérer la réception des messages Register et RegisterStop étant transmis en unicast à destination des RP et des DR du domaine. Il est nécessaire de les identifier et de les transférer à la tâche responsable de leur gestion.

La gestion du routage multicast est quant à elle plus complexe. Il est nécessaire de gérer une TIB contenant les informations de tous les arbres multicast actifs passant par ce routeur. Chaque message multicast entrant doit être analysé et transmis à la tâche qu'il convient. S'il s'agit d'un simple message multicast provenant d'une source, celui-

ci est géré par la tâche « Transit ». Celle-ci retransmet ce message sur toutes les interfaces nécessaires et ce, à l'aide des informations fournies par la TIB. Pour ce faire, il est nécessaire de gérer les arbres. Ceci est réalisé par l'intermédiaire des tâches « Join » et « Prune » auxquels sont transmis les messages correspondants en provenance d'un routeur en aval de l'arbre. A la réception de l'un de ces messages, la fonction met à jour la TIB et, si besoin, réexpédie un autre message en amont.

De plus, la méthode « Transit » doit vérifier si le routeur agit en temps que DR à l'arrivée d'un message en provenance d'un LAN directement connecté. Dans l'affirmative, il doit nécessairement gérer la transmission de message Register vers le RP du groupe et par la même occasion, la gestion des messages Register Stop (méthode « Register »). Le DR doit également prendre en charge les demandes de réception ou de fin de réception d'un flux multicast (méthode « IGMP »).

Chacune de ses différentes tâches doit être en mesure de transmettre l'un ou l'autre message en amont d'un arbre (Soit en direction du RP du groupe, soit en direction d'une source). Pour ce faire, il doit lui être possible d'interroger la table de routage unicast et ce, afin de déterminer l'interface menant à la bonne direction. Il n'est pas nécessaire d'implémenter une MRIB dans le simulateur. L'interrogation directe de la table de routage unicast est suffisante.

L'ensemble des tâches nécessaires étant maintenant défini, voyons comment les intégrer au simulateur. Commençons par l'objet station. Son mode de gestion des messages entrants et sortants étant le plus simple, nous intégrerons directement les deux méthodes dans l'objet Element (méthode « in » et « out »). Par défaut, la méthode « out » retransmettra un message sur toutes les interfaces connectées. Celles-ci seront surchargées par les objets (ce sera le cas de l'objet routeur) le nécessitant. Par contre, les méthodes permettant de faire agir la station en tant que receveur d'un groupe (méthodes « addRecever » et « removeRecever ») ou émetteur (méthodes « addSource » et « removeSource ») seront bien implémentées directement dans l'objet Station

Pour le routeur, la méthode « in » sera surchargée. Elle sera responsable de la réception des messages en provenance de toutes ses interfaces, du contrôle de ceux-ci et de leur transmission à la méthode spécialisée chargée de son traitement. N'ayant pas besoin d'un traitement spécialisé des messages unicast (excepté pour la réexpédition de ceux-ci), cette méthode encapsulera les fonctions « Transmission », « Unicast.in » et « PimSM.in ». Nous n'utiliserons donc l'objet Unicast que pour la gestion de la table de routage unicast. Le routeur possédant plus d'une sortie, il est également nécessaire de lui ajouter une deuxième méthode « out » spécialisée pour l'émission d'un message sur une interface précise.

La gestion réelle des tables de routage unicast n'est pas une mince affaire. Pour simplifier celle-ci, elle sera simulée directement par la méthode « route » de l'objet Domaine. Celui-ci étant directement connecté à tous les objets routeurs ainsi qu'à tous les objets connexions, il lui sera possible de créer un graphe et d'y appliquer un algorithme de recherche de chemins minimaux entre chaque point de ce graphe. Cette méthode utilisera directement la méthode « setRoute » de l'objet unicast de chaque routeur afin de maintenir à jour la table de routage de ceux-ci. Nous retrouverons donc

dans chacun de ceux-ci une table contenant l'interface de sortie menant de ce routeur, à chaque destination possible du réseau. Les autres méthodes du routeur pourront interroger celle-ci via la méthode « getRoute ».

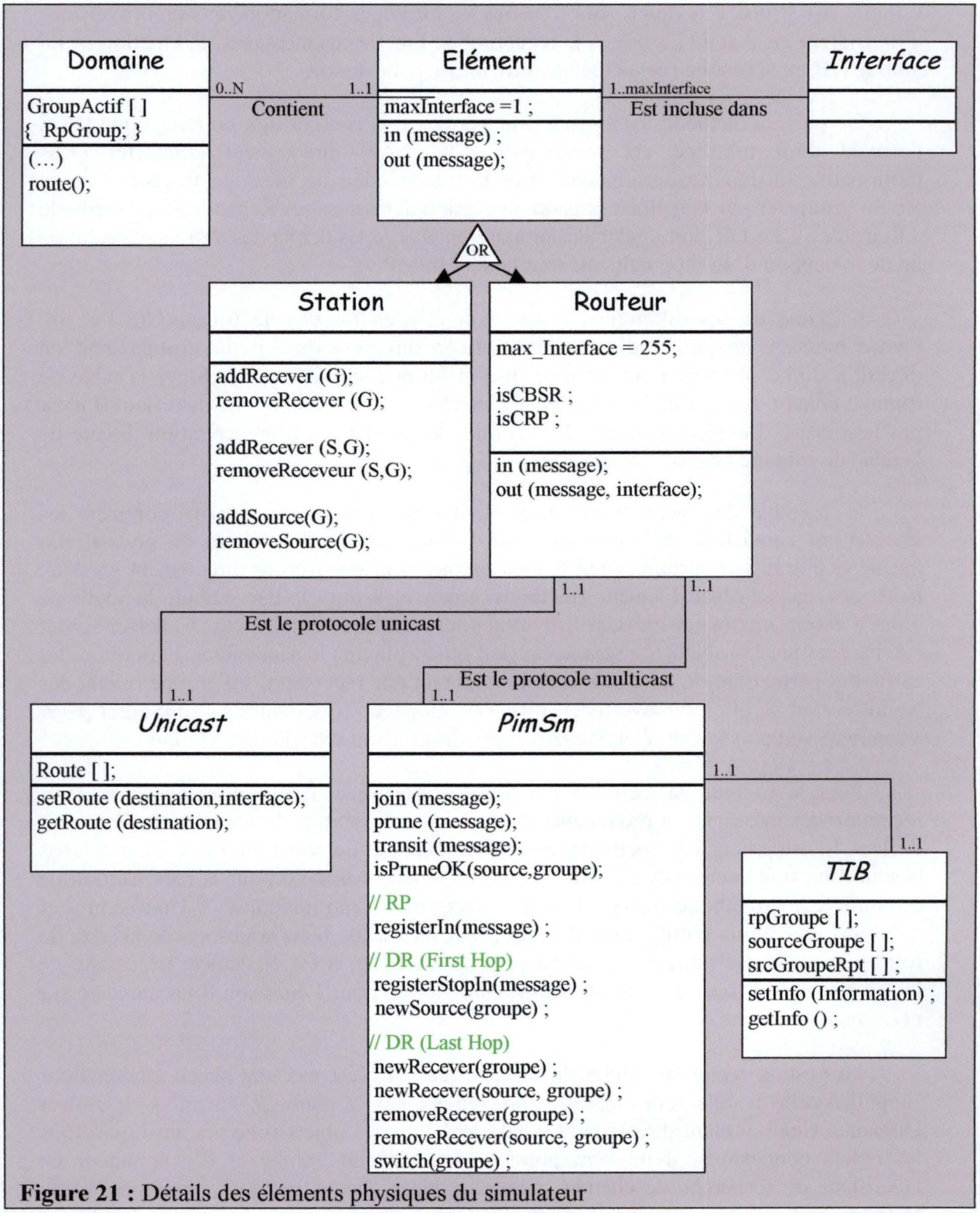


Figure 21 : Détails des éléments physiques du simulateur

La gestion des messages 'Assert' est également lourde. Nous la simulerons en intégrant également une table de routage aux LAN. Cette table sera gérée de la même manière que la table des routeurs (Calculée et configurée par l'objet Domaine).

Les méthodes propres à la gestion multicast seront encapsulées dans l'objet PimSM. Chaque routeur possédant sa propre instance de cet objet, la méthode « routeur.in » à la possibilité de transférer directement les messages multicast aux méthodes de cet objet PimSM. Nous y retrouverons donc les méthodes « Join », « prune » et « transit ». Mais également les méthodes plus spécialisées permettant au routeur d'œuvrer en temps que RP, de DR pour la gestion des sources (First Hop) et des receveurs (Last Hop).

A cet objet PimSm, nous associerons également un objet TIB responsable de la gestion de la « Tree Information Base » reprenant toutes les informations nécessaires à la gestion des flux multicast au niveau de ce routeur. Les méthodes de l'objet PimSm pourront modifier ces information à l'aide des différentes méthodes « setInfo » et obtenir la configuration actuelle d'un flux à l'aide des différentes versions de la méthode « getInfo ».

Nous avons choisi de ne pas implémenter de timer.Cependant, ceux-ci jouent un rôle très important pour la gestion de la prise en compte d'un message Prune en provenance d'un LAN. Dans la réalité, sa prise en compte est temporisée dans le but de laisser le temps à un autre routeur de contrer la demande. Ce mécanisme devra donc être simulé (méthode « isPruneOK »)

5.1.3. Les messages

Une bonne implémentation de la gestion des messages est importante pour assurer un fonctionnement suffisamment fluide du simulateur. Voyons de plus prêt quels sont les besoins réels de celui-ci. Le premier point remarquable est qu'il n'est pas nécessaire d'implémenter une réplique exacte des messages émis dans un domaine réel. Ceux-ci comportent une grande quantité de champs qui ne nous sont pas d'une grande utilité.

Dans le simulateur, un message est simplement émis par une source et expédier vers une destination. Ce message peut posséder un contenu mais il n'en a pas l'obligation. Ce contenu pouvant être une simple suite de caractère dont la signification ne nous importe pas. Mais il peut également s'agir d'un autre message encaplusé dans celui-ci. De plus, chaque message est soit du type "Unicast", soit du type "Multicast". Il est également possible de définir un sous type précisant son utilisation.

Les différents messages émis par une station possèdent un contenu quelconque et sont soit du type "unicast", soit du type "Multicast". Ils sont représentés par l'objet "Message"(cf. figure 22). Celui-ci est suffisant pour représenter tous les messages émis par les stations. Il serait possible d'y ajouter une série de champs permettant d'intégrer tous les autres types de message dans ce même objet. Il s'agit principalement des messages émis par les routeurs et servant à assurer le fonctionnement du protocole. Il est cependant plus propre et plus simple pour la gestion de ceux-ci d'utiliser des spécialisations de cet objet de base.

Quels sont ces messages et les quels devons nous implémenter ? Pour la gestion des flux unicast, un router doit émettre un grand nombre de message permettant la gestion des tables de routage. Cependant, nous simulons cette tâche directement dans l'objet "Domaine" et pouvons nous passer d'implémenter ces messages.

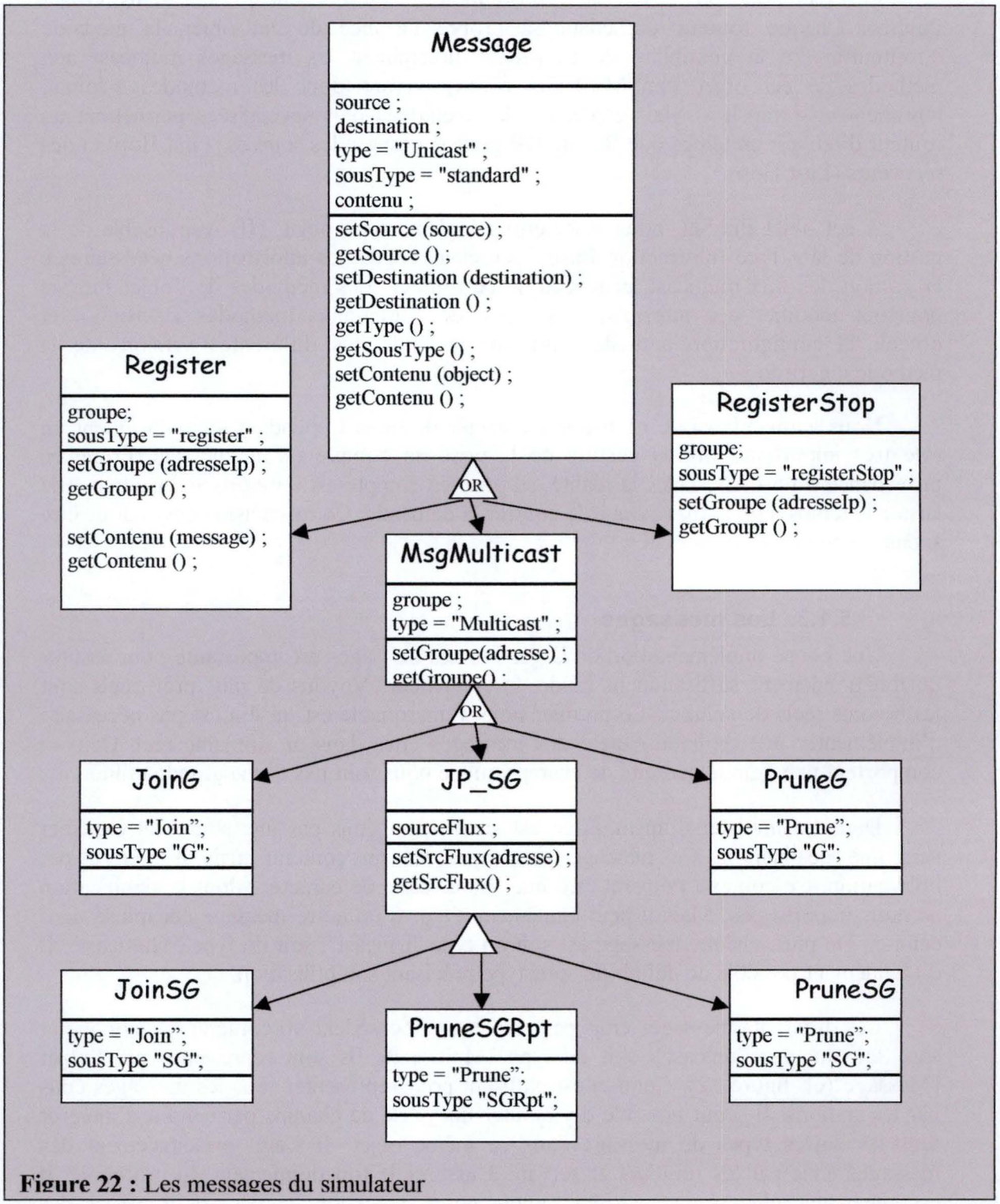


Figure 22 : Les messages du simulateur

Les seuls messages émis en mode unicast nécessaires à la gestion du simulateur sont les messages "Register" émis par les DR et les "RegisterStop" émis par le RP d'un groupe à la réception d'un message "Register". Ces deux types de messages sont implémentés par une extension directe de l'objet "Message" auquel ont ajoute la gestion du groupe multicast. De plus, le contenu du message "Register" doit-être le message multicast originaire de la station émettrice. Les méthodes ("setContenu" et "getContenu") nécessaires à la gestion de celui-ci sont donc à implémenter dans cet objet.

PIM SM nécessite également l'utilisation du message "Hello" ayant pour but de faire connaître le routeur PIM à ses voisins et de pouvoir gérer l'élection des DR. Cependant, ces tâches seront également simulées par l'utilisation de l'objet "Domaine" et par la méthode "setDr" de l'objet Lan. Il en est de même pour les messages "Assert" également gérés par la méthode de l'objet Lan. Ainsi que les messages "Bootstrap" et "Candidate-RP" gérés par les méthodes de l'objet "Domaine".

Il n'en est pas de même pour les messages "Join/Prune". Ceux-ci sont les principaux messages utilisés par le protocole. Dans une implémentation fonctionnelle de celui-ci, un routeur regroupe les différents messages Join et messages Prune en un seul message "Join/Prune". Ceci dans le but d'économiser le temps et la bande passante nécessaire à la transmission de ces derniers. Nous ne sommes cependant pas soumis aux mêmes contraintes. Au contraire, le travail nécessaire pour grouper et dissocier ces messages est une tâche relativement lourde et inutile. C'est pour cette raison que nous les gèrerons séparément.

Ces messages multicast concernent tous un groupe multicast. Nous étendrons donc l'objet "Message" en un objet spécialisé (objet "MsgMulticast") pour la gestion des différents Join et Prune. Les méthodes nécessaires à la gestion du groupe seront également incluses dans cet objet. Les Join et Prune (*,G) seront gérés par une extension directe (objet "JoinG" et "PruneG") de cet objet.

Les autres types de Join et Prune concernent non seulement un groupe, mais également une source spécifique. Nous utiliserons pour ceux-ci une extension de l'objet "JP_G" possédant les méthodes nécessaires à la gestion de cette source. Les messages Join et Prune (S,G) émis en direction de la source S ainsi que les Join et Prune (S,G,rpt) émis en direction de la racinne du groupe seront implémentés par une extension directe de cet objet (objet "JoinSG", "PruneSG" et "PruneSGrpt" possédant les types et sous type appropriés).

5.2. L'analyse de l'interface utilisateur

Nous venons de définir dans les chapitres précédents les différents objets de l'application nécessaire à la simulation du protocole. Les différents noms de méthodes ainsi que les paramètres associés à celles-ci sont, à ce stade, fournis plus à titre d'exemple afin de préciser le fonctionnement de l'objet que d'obligation. Ces méthodes, paramètres et interactions entre les objets seront précisés lors de l'analyse conceptuelle détaillée. A cette occasion, il sera également nécessaire d'ajouter à ces différents objets les méthodes permettant leur emploi par l'interface. Il conviendra par exemple, d'ajouter à chacun d'eux une méthode permettant l'affichage de leur état ainsi qu'une

méthode permettant la sauvegarde et la récupération de l'instance de l'objet. Cependant, avant de réaliser cette analyse détaillée, il est nécessaire de se pencher sur la définition de l'interface et de ses besoins.

5.2.1. Fonctionnalités de l'interface

L'étude des stéréotypes des utilisateurs et du contexte de travail nous permet de définir le type d'interface nécessaire ainsi que des moyens d'interactions à développer. Il s'agit ici d'une interface « Homme – Simulateur ». Son rôle consiste à transmettre les commandes de l'utilisateur vers le simulateur et de retourner à l'utilisateur les modifications des éléments du simulateurs.

Dans ce système, il est important d'utiliser une représentation graphique claire et précise du domaine. Dans celle-ci, les différents éléments seront distinctement représentés par l'utilisation de graphisme explicite. De plus, pour chacune des représentations d'un élément, les principaux paramètres de sa configuration devront également être représentés graphiquement. Il est également nécessaire de représenter graphiquement les différentes connexions reliant ces objets entre eux. Une description complète de la configuration d'un élément doit être disponible sur demande de l'utilisateur.

L'objectif du simulateur est la gestion simultanée de différents groupe multicast. Pour ce faire, l'utilisation d'une couleur configurable pour chacun des groupes actifs simplifie grandement la perception de l'état de chaque groupe. Ceci permettra une représentation simple et efficace de l'état des arbres de diffusion de chaque groupe, des différents RP et également l'identification des sources et des receveurs.

- Pour chaque groupe actif :
  - Couleur du groupe (utilisé pour définir l'appartenance d'un élément)

- Par station :
  - Liste des groupes vers lesquels la station émet du trafic
  - Liste des flux réceptionnés par la station

- Par routeur :
  - Liste des groupes pour lesquels la station œuvre comme RP

- Par connexion :
  - Liste des flux traversant la connexion.

Figure 23 : Attributs représentés graphiquement ou permettant celle-ci.

Au cours de l'exécution de la simulation, outre les modifications de la représentation graphique du domaine, il est nécessaire de signaler à l'utilisateur de manière détaillée les évènements (cf. figure 24) se déroulant. Ceci sera réalisé à l'aide d'une deuxième fenêtre textuelle. De plus, chaque utilisateur ayant des besoins différents, il sera possible de configurer le simulateur afin que celui-ci ne notifie que les

types d'actions désirées par l'utilisateur. Ceci permettra par exemple à un enseignant de focaliser l'attention sur la gestion de l'un ou l'autre événement et ne pas perturber la compréhension de son auditoire par les événements qu'il estime secondaires.

- Ajout et Suppression d'un LAN
- Ajout et Suppression d'une Station
- Ajout et Suppression d'un routeur
- Configuration CRSR d'un routeur
- Configuration CRP d'un routeur
- Election du BSR
- Election des RP
- Election des DR
- Ajout et Suppression d'un émetteur
- Ajout et Suppression d'un receveur
- Emission et Réception d'un "Register"
- Emission et Réception d'un "RegisterStop"
- Emission et Réception d'un "Join"
- Emission et Réception d'un "Prune"
- Emission et Réception d'un mes. Multicast
- Emission et Réception d'un mes. Unicast
- Modification d'une TIB
- Modification de l'arbre d'un flux

Figure 24 : Liste des événements possibles en provenance du simulateur

- **Menu contextuel d'un Domaine :**
  - Activation et Désactivation de la simulation
  - Affichage de la configuration détaillée du domaine
  - Assistance (Qu'est-ce qu'un domaine)
- **Menu contextuel d'une Station :**
  - Activation et Désactivation de l'émission vers un groupe
  - Emission d'un message vers un groupe
  - Activation et Désactivation de la réception d'un flux (*,G)
  - Activation et Désactivation de la réception d'un flux (S,G)
  - Affichage de la configuration détaillée de la station
  - Suppression de la station
  - Assistance (Qu'est-ce qu'une station)
- **Menu contextuel d'un Routeur :**
  - Activation et Désactivation de l'état CBSR
  - Activation et Désactivation de l'état CRP
  - Affichage de la configuration détaillée du routeur
  - Suppression du routeur
  - Assistance (Qu'est-ce qu'un routeur)
- **Menu contextuel d'un Objet connexion :**
  - Suppression de la connexion
  - Affichage de la configuration détaillée de la connexion
  - Assistance (Qu'est-ce qu'une connexion)

Figure 25 : Menus contextuels

Afin de simplifier la gestion de l'interface, une grande place sera donnée à l'utilisation de la souris. C'est pourquoi, outre la barre de menu traditionnel, une barre d'outil sera également développée. De plus, une série de menu contextuel sera également ajoutée à l'interface (cf. figure 25). Nous avons vu qu'il était important

d'apporter à l'utilisateur une assistance efficace. Outre l'habituel menu « Aide » du menu principal, il est intéressant d'ajouter une assistance contextuelle à chacun des éléments du domaine.

5.2.2. Structuration de la tâche

Après avoir abordé l'aspect général de l'interface et ses moyens d'interaction, il est temps de se pencher sur les différentes fonctionnalités que celle-ci doit fournir à l'utilisateur afin de lui permettre la réalisation de la tâche principale du simulateur.

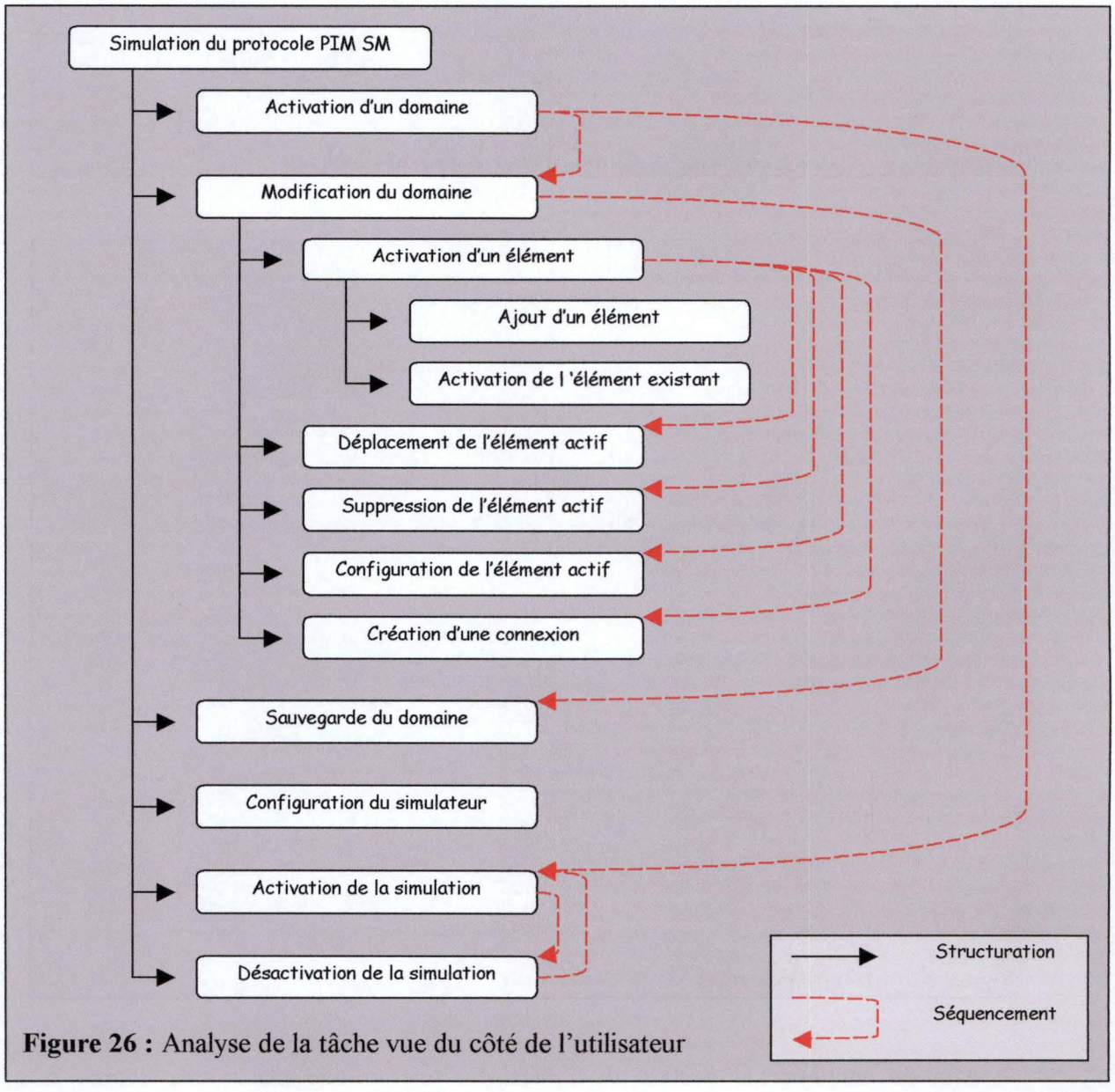


Figure 26 : Analyse de la tâche vue du côté de l'utilisateur

Découpons cette tâche principale qu'est la "simulation du protocole PIM SM" en sous tâches (cf. figure 26). La simulation du protocole est effectuée à partir de la

représentation d'un domaine. La première sous-tâche nécessaire est donc l'activation d'un domaine. Pour ce faire, l'utilisateur a le choix entre demander l'ouverture d'un fichier (contenant un domaine préalablement sauvegardé) qu'il aura sélectionné ou la création d'un nouveau domaine vierge de tout élément (cf. figure 27).

Lorsqu'un domaine a été activé, l'utilisateur a la possibilité de modifier celui-ci (sous-tâche "Modification du domaine"). Cette sous-tâche commence nécessairement par l'activation d'un élément. C'est-à-dire l'ajout d'un nouvel élément (cf. figure 28) qui est automatiquement affiché par l'interface et activé ou la sélection d'un élément déjà existant. Signalons que, contrairement à la sélection d'un élément existant, l'ajout d'un nouvel élément représente déjà une modification. Il est ensuite possible à l'utilisateur de déplacer celui-ci, de le supprimer, de modifier sa configuration (celle-ci dépendant de l'élément activé) ou d'établir une connexion avec un autre élément du domaine.

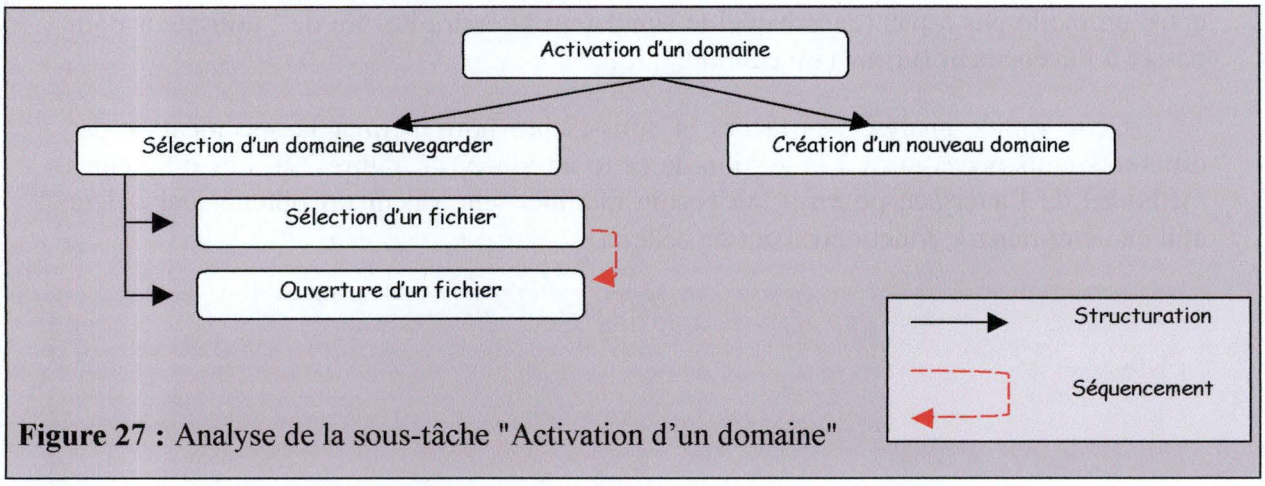


Figure 27 : Analyse de la sous-tâche "Activation d'un domaine"

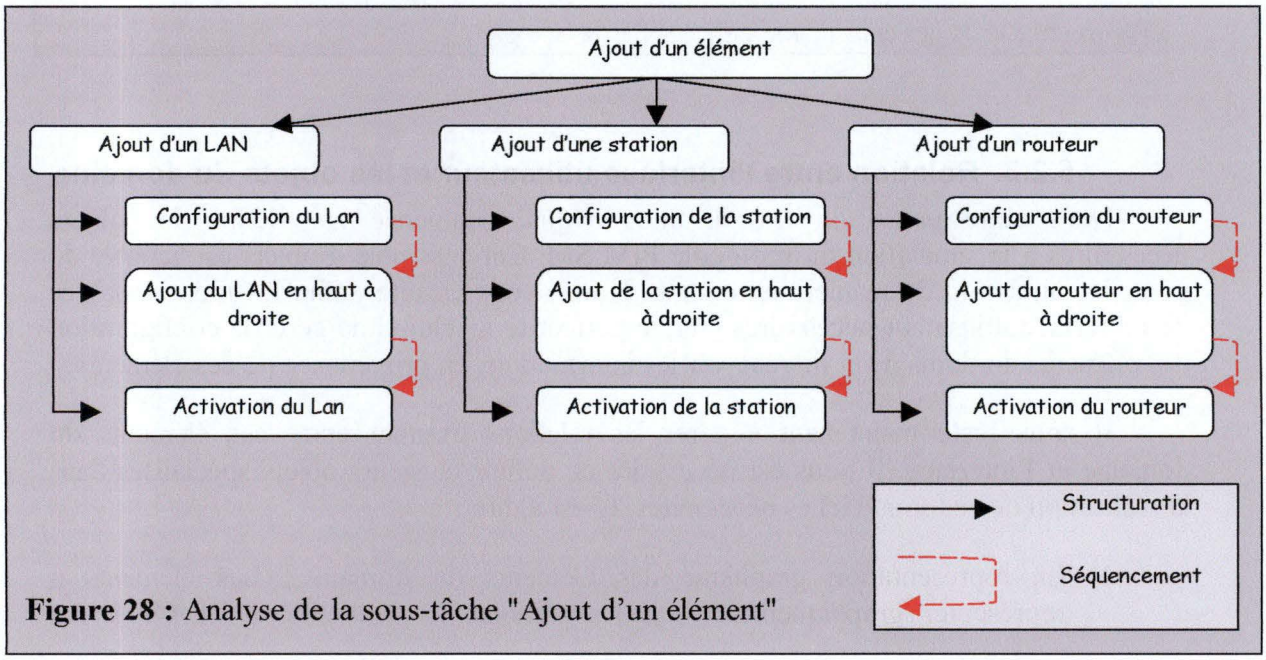


Figure 28 : Analyse de la sous-tâche "Ajout d'un élément"

A la suite de la modification du domaine actif l'utilisateur à la possibilité de sauvegarder celui-ci. Cette sous-tâche devra encore être décomposée en "Sélection d'un nom" suivit de l'enregistrement à proprement dit. La sous-tâche "Sélection d'un nom" étant optionnelle si le domaine actif possède déjà un nom.

Au moment de son choix, l'utilisateur a également la possibilité de modifier la configuration du mode de fonctionnement du simulateur (sous-tâche "Configuration du simulateur"). Cette configuration consiste principalement à la sélection des événements en provenance du simulateur qui intéressent l'utilisateur et à la configuration des couleurs utilisées pour chacun des groupes actifs.

L'utilisateur a également la possibilité d'activer la simulation lorsqu'un domaine est actif. Il peut ensuite à sa guise passer du mode actif au mode inactif. Ces actions étant respectivement réalisées par l'intermédiaire des sous-tâches "Activation de la simulation" et "Désactivation de la simulation". Il lui sera également possible de choisir entre un mode pas à pas (dans lequel le simulateur attendra l'ordre de l'utilisateur pour passer à l'événement suivant) ou continu.

Cette rapide analyse des tâches et sous-tâches nous permet la spécification des différents états nécessaires à la gestion de cette interface (cf. figure 29). Les différentes méthodes de l'interface pourront au besoin modifier leur valeur ou obtenir leur valeur afin de déterminer le fonctionnement de celle-ci.

ETAT	Valeurs possibles de l'état
- Domaine Actif	Oui / Non
- Elément Actif	Null / objet Elément
- Domaine Modifié	Oui / Non
- Nom du fichier	Null / objet String
- Simulation Active	Oui / Non

Figure 29 : Liste des états de l'interface

5.2.3. Relation entre l'interface utilisateur et les objets du domaine

Dans cette partie du travail, nous avons commencé par définir les objets nécessaires à la simulation du protocole PIM SM. Cet ensemble d'objets est capable de gérer la simulation de manière autonome. Nous avons ensuite défini les fonctionnalités de l'interface utilisateur nécessaires afin de permettre à celui-ci de gérer la configuration des éléments du domaine et de recevoir les informations en provenance de ces éléments.

Il nous reste maintenant à gérer les relations existant entre ces éléments du domaine et l'interface. Il nous est nécessaire de définir plusieurs objets spécialisés dans la réalisation de ces sous-tâches nécessaires. C'est à dire :

1) La représentation graphique des éléments du domaine : Les éléments à représenter graphiquement sont les stations, les routeurs, les LAN et les

connexions PàP. Une classe (classe « ObjGraphique ») sera spécialisée afin que chaque instance de celle-ci encapsule un élément graphique. Cette instance gérera :

- L'affichage de la représentation graphique de l'élément du domaine
- L'affichage de la configuration de l'élément du domaine
- La prise en charge des actions de l'utilisateur sur cette représentation

2) La représentation graphique des interfaces : Les éléments actifs sont reliés aux connexions par l'intermédiaire d'interface. Il est également nécessaire de les représenter graphiquement. Pour ce faire, une classe sera également spécialisée (classe « ObjInterfaceGraphique »). Chaque instance de celle-ci encapsulera une interface du domaine et gérera :

   - L'affichage de la représentation graphique de l'interface
   - L'affichage de la configuration de l'interface
   - La prise en charge des actions de l'utilisateur sur cette représentation

3) La représentation des évènements du domaine : Les instances des classes formant le domaine travaillent de manière autonome. Il est cependant nécessaire de leur fournir un moyen de communiquer avec l'interface. Pour ce faire, nous utiliserons une classe spécialisée dans la gestion des évènements (classe « GestionEvenements »). Cette classe, configurée statiquement à l'initialisation de l'interface, permettra à chaque objet du domaine de signaler un événement se produisant à son niveau. Elle sera responsable de :

   - La gestion des types d'évènements à dispatcher
   - La gestion du dispatching ou non d'un événement signalé par le domaine
   - La gestion du mode pas-à-pas ou continu de la simulation

6. Implémentation

Dans le chapitre précédant, nous avons réalisé la découpe en objet du domaine et de l'interface utilisateur. Nous avons également défini les rôles de ces différents objets. Dans cette partie du travail, nous allons détailler les choix d'implémentations réalisés afin de permettre la simulation.

La taille de ce travail ne nous permettant pas de passer en revue la totalité des méthodes employées dans tous les objets, nous nous limiterons aux plus importantes et significatives.

6.1. Gestion des adresses IP

Dans le simulateur, la classe «AdresseIp» est spécialisée pour la gestion des types d'adresses nécessaires. Ces adresses utilisées sont de deux types :

- les adresses IP des interfaces
- les adresses IP multicast

Chaque connexion (LAN et ligne PàP) possède une adresse réseau unique. La classe «AdresseIp» est responsable de l'attribution et de la gestion de ces adresses. La première connexion recevra l'adresse 10.10.1.xx, la deuxième 10.10.2.xx et ainsi de suite à l'aide de la méthode «setNextAdresseReseau()». Au raccordement d'un élément actif à une connexion, une interface est créée. La connexion est responsable de la configuration de l'adresse IP de cette interface. Elle utilise pour ce faire la méthode «setAdresse (adrReseau, numStation)» ou adrReseau est sa propre adresse réseau et numStation un numéro de station géré par la connexion commençant à un et incrémenté à chaque nouvelle liaison.

Chaque groupe multicat actif sur le domaine reçoit une adresse multicast. Cette adresse doit-être unique. C'est également la classe «AdresseIp» qui, par l'intermédiaire de la méthode «setNextAdresseMulticast()», se charge de la gestion de celles-ci. Le

premier groupe multicast recevra l'adresse 225.10.10.1, le deuxième 225.10.10.2 et ainsi de suite.

La classe «AdresseIp» est également responsable de la comparaison entre adresse (méthodes «compareTo(AdresseIp)», «compareToLan(AdresseIp)» et «isLower(AdresseIp)»). Toutes ces méthodes, bien qu'améliorables, permettent une gestion des adresses IP suffisante pour le fonctionnement du simulateur.

6.2. Le routage unicast

Il a déjà été défini que le routage sera simulé par l'objet domaine et que celui-ci enregistrera directement les routes aux routeurs du domaine via leur propre méthode "route". Afin de simuler la gestion des "Assert", nous lui ferons également exécuter le même travail pour les LAN et enregistrer le résultat directement au LAN via la méthode "setAssert".

Les éléments du domaine susceptibles d'être à la source ou destination d'un message circulant sur le domaine sont les stations et les routeurs. Cependant, chaque station est reliée à un LAN. Nous réaliserons donc le routage sur base des LAN et des routeurs du domaine.

Le graphe formé par les routeurs et les LAN du domaine est constitué de chemins de poids positif et peu éventuellement comporter des circuits. C'est pourquoi les calculs des routes seront effectués à l'aide d'une implémentation de l'algorithme Moore-Dykstra. Celui-ci travaille sur une matrice comprenant les poids des routes directes entre deux éléments. Nous implémenterons donc la gestion de celle-ci. Cette matrice comprendra les renseignements sur les routes directes partant de chaque routeur ou LAN du domaine et leur destination. Dans cette version du simulateur, nous ne travaillerons pas avec des poids de grandeur variable mais donnerons le même poids à chaque connexion du même type. La prise en charge de connexions de poids variable compliquerait la compréhension du protocole sans apporter énormément au simulateur. Cependant, celle-ci reste facilement implantable.

Nous donnerons un poids de 1 à chaque liaison directe entre un routeur et un LAN et un poids de 2 à chaque liaison entre deux routeurs via une ligne PàP. Ceci permettant de donner la même importance à une liaison entre deux routeurs et ce, qu'elle passe par une ligne PàP ou via un LAN.

A chaque établissement d'une liaison entre un routeur et une connexion, les tables de routage doivent être mises à jour. Un appel à la méthode «route()» est effectué. Voyons cela de plus près :

```
Méthode Route ( )
Début
        PrepareMatriceRoutage()   // Initialisation de la matrice de routage
        Pour tout  I tel que I = élément de la matrice

                initRoutage ( I )   // Initialisation du routage pour l'élément
                executeDijkstra ( I ) // Calcul des routes à partir de cet élément
                clotureRoutage ( I ) // Sauvegarde du résultat dans l'élément
        Fin de Pour tout
Fin
```

Cette méthode principale appelle un ensemble de méthodes secondaires ayant chacune leur rôle. La première permet de préparer la matrice de routage qui sera utilisée pour le calcul des routes de tous les éléments. Celle-ci ne sera pas modifiée durant le traitement de ces éléments.

```
Méthode PrepareMatriceRoutage()
Début
        MatRoutage = NEW MatriceRoutage   // Initialisation de la matrice de routage

        // Traitement des éléments Actif du domaine
        Pour tout  I tel que I = élément actif  du domaine
                Si I instance de Routeur
                        // Ajout d'une nouvelle ligne à la  matrice de routage
                        MatRoutage = MatRoutage + I
                Fin Si
        Fin de Pour tout

        // Traitement des connexions  du domaine
        Pour tout  I tel que I = connexion  du domaine
                Si I instance de LAN
                        // Ajout d'une nouvelle ligne à la  matrice de routage
                        MatRoutage = MatRoutage + I
                Fin Si
        Fin de Pour tout

        // Initialisation des éléments de la matrice
        // ( MatRoutage(X,Y) concerne l'interface menant de l'élément
        // d'indice X dans la matrice à l'élément d'indice Y dans la matrice)
        Pour tout I tel que I = élément de la matrice
                Pour tout J tel que J = élément de la matrice sauf I
                        S'il existe interface entre I et J
                        // Liaison directe  entre un Routeur et un LAN
                          MatRoutage ( I, J ) = PoidsLiaisonLAN  // = à un
                        Sinon S'il existe PàP entre I et J
                        // Liaison entre deux routeurs via une ligne PàP
                          MatRoutage ( I, J ) = PoidsLiaisonPAP  // = à deux
                        Sinon
                          MatRoutage ( I, J ) = null
                        Fin Sinon
                Fin de Pour tout J
        Fin de Pour tout I
Fin
```

La matrice de routage étant complétée, le calcul des routes peut-être réalisé. Pour chaque élément, nous commençons par la réinitialisation de celui-ci ainsi que la préparation des vecteurs permettant le calcul des routes avec cet élément comme racine.

```
Méthode initRoutage( element )
Début
        // Initialisation de l'élément
        Si element Instance de Routeur
                // Réinitialisation de la table de routage du routeur
                Element.unicast = New Unicast
        Sinon
                // Réinitialisation de la gestion des asserts du LAN
                Element.Assert = New Assert
        Fin Sinon

        // initialisation du calcul des routes
        Pour tout I de 1 à Nbr élément de la matrice
                interfaceSortie ( I ) = Null
                distanceTo ( I ) = Null
        Fin de Pour tout
        // initialisation de la racine de l'arbre
```

```
        distanceTo (element) = 0 ;

        // initialisation des nœuds directement joignables à partir de la racine
        Pour tout I de 1 à Nbr élément de la matrice sauf indice de element
                Si MatRoutage (indice de element, I ) Not Null
                        interfaceSortie ( I ) = interface entre reliant les 2 elements
                        distanceTo ( I ) = MatRoutage (indice de element, I )
        Fin de Pour tout
Fin
```

L'initialisation de l'élément traité étant terminée, nous pouvons réalisé le calcul du routage à partir de cet élément.

```
Méthode executeDijkstra ( element )
Début
        AmeliorationTrouvee = true

        Tant que  AmeliorationTrouvee
                AmeliorationTrouvee = false

                // pour tout déjà accessible excepté la racine
                Pour tout I tel que I = élément de la matrice
                                Et distanceTo ( I ) > 0

                        Pour tout J tel que J = élément de la matrice

                                // si les deux éléments sont directement connectés  et que cette
                                // connexion améliore la distance entre la racine et J
                                Si MatRoutage( I, J) Not null
                                Et distanceTo ( J ) < distanceTo ( I ) + MatRoutage( I, J)

                                        AmeliorationTrouvee = true
                                        interfaceSortie ( J ) = interfaceSortie ( I )
                                        distanceTo ( J ) = distanceTo ( I )+MatRoutage( I, J)

                                Fin Si
                        Fin de Pour tout J
                Fin de Pour tout I
        Fin Tant que
Fin
```

Le calcul terminé, il reste à configurer l'élément traité. C'est à dire les lignes de la table de routage pour un routeur ou la direction menant aux « assert winner » des éléments s'il s'agit d'un LAN.

```
Méthode clotureRoutage ( element )
Début
        // Pour chaque élément accessible à partir de l'élément traité
        Pour tout I tel que I = élément de la matrice sauf element
                        Et distanceTo ( I ) > 0

                Si element instance de Routeur
                                // configuration de la route en direction de l'élément
                                element.unicast.setRoute ( I , interfaceSortie ( I ) )
                Sinon // l'élément est un LAN
                                // configuration de la direction de l'assert Winner vers I
                                element.setAssert ( I , interfaceSortie ( I ) )
                Fin Sinon

        Fin de Pour tout
Fin
```

6.3. Election des éléments du domaine multicast

Nous avons déjà spécifié que le mode d'élection du BSR du domaine, des différents RP et des DR sera simulée par le domaine et les LAN. Voyons cela de plus près.

6.3.1. Election du BSR

Contrairement à la réalité, dans le simulateur le rôle du BSR est purement fictif. L'implémentation des messages «Hello» n'étant pas réalisée, le routeur élu comme BSR n'effectue ici aucune action. C'est l'objet domaine qui s'en charge.

A chaque changement d'état du mode « Candidat BSR » d'un routeur du domaine, l'élection du BSR est réalisée (méthode «setBsr()»).

```
Méthode setBsr ( )
Début
        Pour tout I tel que I = élément actif du domaine

                // Si l'élément Actif est un routeur Candidat BSR
                Si I instance de Routeur
                Et I.isCBSR = true

                        // Si aucun BSR n'est actif ou que l'adresse IP de celui-ci est
                        // supérieure à l'adresse IP du routeur traité
                        Si BSR = null
                        Or I.AdresseIp < BSR. AdresseIp

                                BSR = I
                        Fin Si
                Fin Si

        Fin de Pour tout
Fin
```

6.3.2. Election des RP

L'élection des différents RP est normalement calculée par chaque routeur et ce, en appliquant une méthode de hashing sur la liste des RP fournie par le BSR. Dans le simulateur, c'est le domaine qui associe un Candidat RP à chaque groupe multicast actif sur le domaine.

A chaque changement d'état du mode « Candidat RP » d'un routeur du domaine ou à chaque activation d'un groupe multicast, l'élection des RP est réalisée (méthode «setRp()»). A chaque groupe multicast actif est associé un RP. Ceci est réalisé de manière à distribuer équitablement la charge entre les CRP du domaine.

```
Méthode setRp ( )
Début
        crpExiste = False
        grpSélectionné = null
        elementSélectionné = null

        Faire :
                Si tous les groupes actifs sont traités
                        FIN
                Fin Si

                Si tous les éléments actifs ont été parcourus
```

```
                    Si crpExiste = False
                        FIN
                    Sinon
                        Revenir au premier élément actif
                    Fin Sinon
                Fin Si

                Si grpSélectionné = null
                    grpSélectionné = groupe suivant
                Fin Si

                elementSélectionné = élément actif suivant

                // Si l'élément est un routeur Candidat RP
                Si elementSélectionné instance de Routeur
                Et elementSélectionné.isCRP = true

                    grpSélectionné.RP = elementSélectionné
                    grpSélectionné = null
                    crpExiste = True
                Fin Si

            Fin Faire
Fin
```

6.3.3. Election des DR

Dans une implémentation réelle de PIM SM, l'élection des DR est effectuée par les routeurs en se basant sur les messages que ceux-ci s'échangent sur les LAN auxquels ils sont directement connectés. Dans le simulateur, chaque LAN est responsable de l'élection de son DR.

Pour ce faire, à chaque connexion ou déconnexion d'un routeur à un LAN, l'élection du DR est effectuée sur celui-ci (méthode «setDr()»).

```
Méthode setDr ( )
Début
        // Réinitialiser le DR si besoin
        Si drWinner Not Null
            DrWinner.isDr = False
            DrWinner = Null
        Fin Si

        Pour tout I tel que I = interface connectée au LAN

            // Si l'interface est reliée à un routeur
            Si I.élémentActif instance de Routeur

                // si aucun DR sélectionné ou si son adresseIp
                // est plus grande que le routeur traité
                Si drWinner = Null
                Ou I.AdresseIP < DrWinner.AdresseIP

                    // Sélectionné ce routeur comme DR
                    DrWinner = I

                Fin Si
            Fin Si
        Fin Pour tout
Fin
```

6.4. La simulation du protocole IGMP

Bien que ne faisant pas directement partie de PIM SM, le protocole IGMP est indispensable afin de permettre aux stations de signaler au DR du LAN auquel elles

sont connectées les flux multicast qu'elles désirent recevoir ou ne plus recevoir. Outre la nécessité de gérer des timers, une implémentation réelle de ce protocole entraîne une certaine complexité et n'apporte rien de plus à la compréhension de PIM SM.C'est pour cette raison que, dans le simulateur, nous le simulons directement sur le LAN(méthode «igmp()»).

Lorsqu'un utilisateur demande la réception ou la fin de réception d'un flux par une station, celle-ci émet un message IGMP directement sur le LAN. S'il s'agit d'une demande de fin de réception, Le LAN interroge directement les autres stations afin de s'assurer que la demande peut-être prise en compte avant de la transférer au DR.

```
Méthode IGMP ( flux, typeDemande, interfaceEntrée )
        // flux = flux multicast (*,G) ou  (S,G) concerné par la demande
        // typeDemande = réception OU Fin de réception
        // interfaceEntrée = interface reliant la station émettrice de la demande au LAN
Début
            InterfaceDR = null

            // traiter toutes les interfaces du LAN
            Pour tout I tel que I = interface connectée au LAN

                    // Si l'interface mène à une station autre que la station émettrice
                    // et que la demande est une fin de réception
                    Si I.ElementActif instance de Station
                    Et I Not = interfaceEntrée
                    Et typeDemande = Fin de réception

                            // Le flux est-il toujours nécessaire pour cette station ?
                            Si I.ElementActif.fluxRecu(flux) = True
                                // La demande ne doit pas être traitée
                                FIN TRAITEMENT

                    Fin Si

                    Si I.isDR = True
                            InterfaceDR = I

            Fin Pour tout

            // La demande (réception ou fin de réception) doit être traitée

            // Si un Dr existe
            Si InterfaceDR Not null
                    //Transférer la demande au DR
                    InterfaceDR. ElementActif.IGMP (flux, typeDemande)
Fin
```

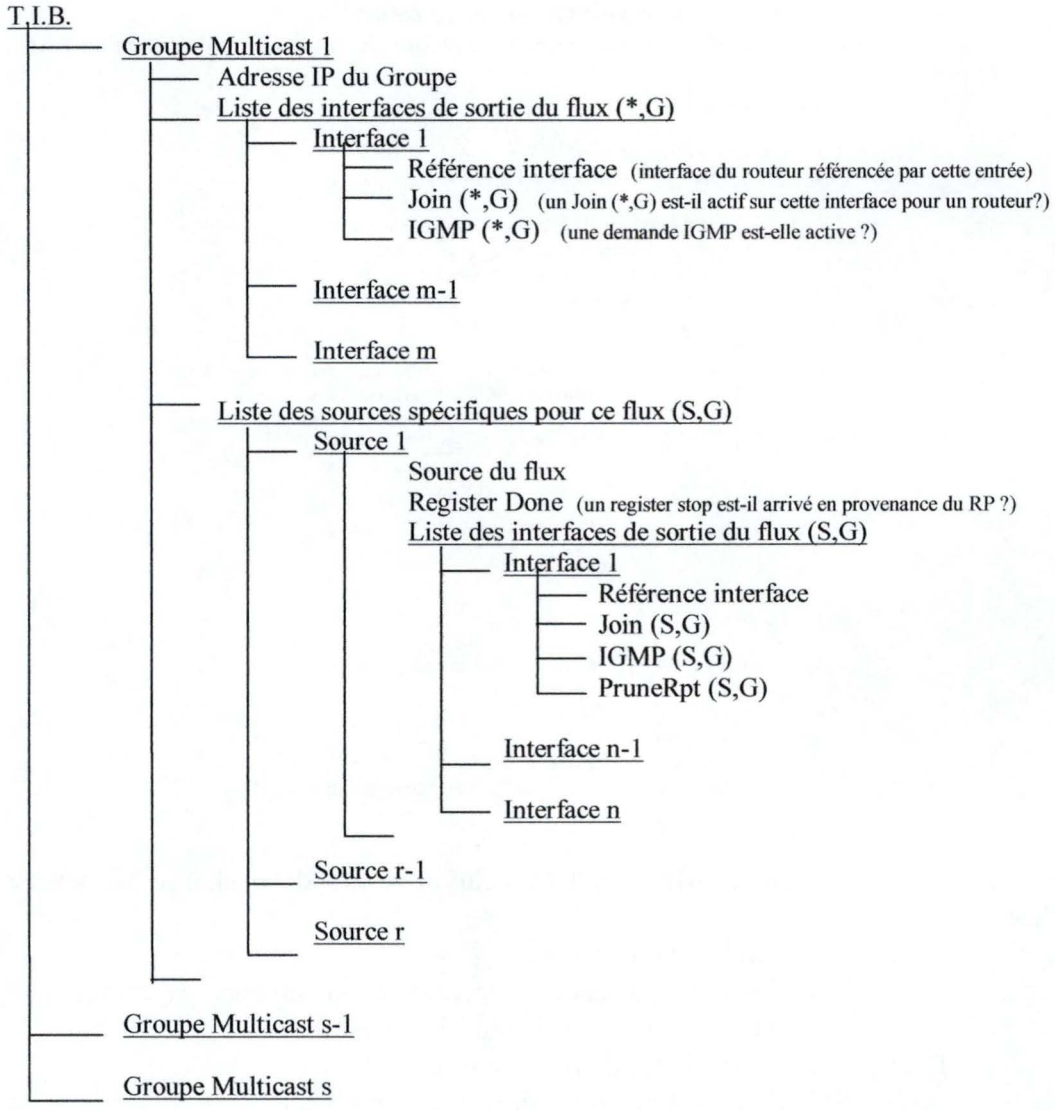
Lorsque la demande arrive au DR, celui-ci la prend en charge silencieusement. C'est à dire que :

- Pour une demande de réception :
  - Si le flux n'est pas encore réceptionné, la demande est traitée
  - Si le flux est déjà réceptionné, rien n'est fait.
- Pour une demande de fin de réception :
  - Si le flux n'est pas réceptionné, rien n'est fait
  - Si le flux est réceptionné, la demande est traitée.

6.5. La TIB

Nous avons déjà vu que la base d'information des arbres (TIB) permet de centraliser toutes les informations nécessaires à un routeur PIM SM pour sa gestion du routage multicast. Chaque routeur possède sa propre TIB. Cette base est gérée par le protocole PIM SM sur base des différents messages qu'il réceptionne. Nous étudierons cela plus en détail dans le point suivant. Pour l'instant, étudions la structure de cette base.

Dans le simulateur, cette table est initialement vide et construite au fur et à mesure des besoins. Bien que simplifiée par rapport aux besoins d'une implémentation réelle, Cette structure reste relativement compliquée. La voici



Les informations sont classées par groupe multicast. Pour Chaque groupe actif sur le routeur, nous retrouvons (outre l'adresse du groupe) une liste des interfaces sur lesquelles un Join(*,G) reçu d'un routeur en aval est actif (C'est à dire qu'il n'a pas été contré par un Prune) Et / Ou une demande IGMP en provenance d'un LAN pour lequel

ce routeur œuvre en temps que DR. Nous retrouvons également une liste des flux (S,G) spécifiques à une source.

Dans cette liste, outre la référence de la source, nous retrouvons un booléen permettant au DR de notifier que le flux en provenance d'une source située sur un de ses LAN n'est plus en mode Register. Nous retrouvons également une liste des interfaces contenant les informations Join et IGMP spécifiques à ce flux (S,G) ainsi qu'un indicateur signalant qu'un message PruneRPT est actif sur cette interface.

L'implémentation de cette structure est réalisée via l'utilisation de cinq classes spécialisées.

1) La classe TIB responsable de l'initialisation de la table et contenant une liste d'instances GrpTIB.

2) La classe GrpTIB responsable de la gestion des informations d'un groupe et contenant une liste d'instances GrpIntTIB et une liste d'instances SrcGrpTIB

3) La classe GrpIntTIB responsable de la gestion des informations spécifiques à une interface pour un flux (*,G) auquel elle appartient

4) La classe SrcGrpTIB responsable de la gestion des informations spécifiques à une source pour le groupe auquel elle appartient et contenant une liste d'instances SrcGrpIntTIB.

5) La classe SrcGrpIntTIB responsable de la gestion des informations spécifiques à une interface pour un flux (S,G) auquel elle appartient

Ces différentes classes ne sont en réalité que des conteneurs d'informations. Il appartient aux méthodes de la classe PimSM de s'occuper de la gestion de ces informations (Ajout, Modification et Suppression).

6.6. L'objet de PimSM

Après avoir abordé les principales fonctionnalités permettant le fonctionnement du réseau sur lequel le protocole PIM SM doit être simulé, il est temps de s'intéresser à celui-ci. La majorité des documents disponibles traitant de PIM SM nous décrivent les différents types de messages du protocole, ainsi que les réactions des routeurs à l'arrivée de ces messages. Ils nous fournissent l'état d'un routeur avant l'arrivée d'un message, les différents messages que le routeur doit émettre lors du traitement de celui-ci et l'état du routeur après le traitement du message. Pour le simulateur, il nous reste à implémenter ces actions dans la classe PimSM.

Pour ce faire, nous commencerons par analyser la marche à suivre à la réception des différents messages Join et Prune en provenance d'un autre routeur. Bien qu'implémentée dans la classe PimSM (méthode « IGMP »), la gestion d'une demande IGMP étant similaire à la gestion des Join et Prune (l'unique différence étant qu'elle

travaille sur les indicateurs IGMP (*,G) et IGMP (S,G) de la TIB), elle ne sera pas détaillée ici.

A la réception de l'un de ces messages, il est généralement nécessaire d'émettre un message similaire en amont de l'arbre. Dans le simulateur, le traitement de ce besoin est centralisé dans une méthode unique (méthode « sendJoinPrune »)que nous détaillerons.

Il nous restera à analyser les différentes méthodes permettant la gestion de la réception d'un message multicast, l'analyse de la TIB afin de créer la liste des interfaces par ou ce message doit-être retransmis et la retransmission du message par ces interfaces. Dans le cas ou le routeur est DR, nous devons également gérer l'émission et la réception de messages Register et RegisterStop.

6.6.1. La réception d'un message Join

Les messages Join sont de deux types :
- les messages Join (*,G)
- les messages Join (S,G)

A l'arrivée d'un de ces messages dans le routeur, celui-ci examine son type et le dispatche à l'une des deux méthodes spécialisées (méthodes « JoinG » et « JoinSG ») de son instance de PimSM.

```
Méthode JoinG ( interfaceEntrée, messageJoinG)
        // interfaceEntrée  = interface d'entrée du message
        // messageJoinG = message Join (*,G) traité
Début
        NeedJoin = False
        Si TIB.Grp(G).ListeInterfacesDeSortie = null

                // Créer l'entrée dans la TIB
                TIB.Grp(G).ListeInterfacesSortie = Liste Vide

                // Un Join sera nécessaire
                NeedJoin = True
        Fin Si

        // Notifié l'arrivée du Join (*,G) via cette interface dans la TIB
        TIB.Grp(G).ListeInterfacesSortie.interface(interfaceEntrée).Join = True

        Si NeedJoin = True

                // Traitement de l'émission des Joins en amont
                sendJoinPrune ( flux(*,G), "Join")
        Fin Si
Fin


Méthode JoinSG ( interfaceEntrée, messageJoinSG)
        // interfaceEntrée  = interface d'entrée du message
        // messageJoinG = message Join (S,G) traité
Début
        NeedJoin = False
        Si TIB.Grp(G).source(S).ListeInterfacesSortie = null

                // Créer l'entrée dans la TIB
```

```
                TIB.Grp(G).source(S).ListeInterfacesDeSortie = Liste Vide

                // Un Join sera nécessaire
                NeedJoin = True
        Fin Si

        // Notifié l'arrivée du Join (*,G) via cette interface dans la TIB
        TIB.Grp(G).source(S).ListeInterfacesSortie.interface(interfaceEntrée).Join = True

        Si NeedJoin = True

                // Traitement de l'émission des Joins en amont
                sendJoinPrune ( flux(*,G), "Join")
        Fin Si
Fin
```

6.6.2. La simulation du timer « Join-Prune Override Interval »

A la réception d'un message Prune, un routeur PIM SM ne le prend pas immédiatement en compte. Le routeur démarre un timer « Join-Prune Override Interval » et ne prend en compte le Prune qu'à l'expiration de celui-ci. Ceci dans le but de laisser à un hypothétique routeur connecté sur le même LAN et nécessitant toujours la réception de ce flux de pouvoir contrer ce Prune en émettant un nouveau Join pour le même flux.

Le choix de ne pas implémenter dans le simulateur la gestion des timers nous oblige à simuler celui-ci. Pour ce faire, à l'arrivée d'un message Prune, nous utilisons la méthode « isPruneOK » de la classe PimSM. Celle-ci utilisera la méthode « verificationPruneOK » de la classe LAN pour vérifier que les autres routeurs connectés à ce LAN n'ont plus besoin du flux traité et ce, par l'intermédiaire de leur méthode «PimSM.isFluxNécessaire ». Si la prise en compte du message Prune peut-être réalisée, cette méthode retourne True sinon False.

```
Méthode PimSM.isPruneOK ( S, G, RPT,   interfaceEntrée)
        // S = Source du flux concerné par le prune ( = null pour un flux (*,G) )
        // G = Groupe du flux concerné par le prune
        // RPT = booléen utilisé dans le cas d'un flux(S,G) signalant un prune (S, G, Rpt)
        // interfaceEntrée  = interface d'entrée du message Prune
Début
        // Si l'interface est connectée à un LAN
        Si interfaceEntrée.Connexion instance de LAN

            // Passer la vérification au Lan
            Return interfaceEntrée.Connexion.vérificationPruneOK(S, G, RPT, interfaceEntrée)

        Fin Si

        // Sur une liaison PàP, le prune peut toujours être pris en compte directement
        Return True
Fin


Méthode Lan.vérificationPruneOK ( S, G, RPT,   interfaceEntrée)
Début
        // Pour toute interface du LAN menant à un routeur
        // autre que le routeur effectuant la demande
        Pour tout I interface du LAN
        Tel que I.ElementActif instance de Routeur et I Not = interfaceEntrée
```

```
            // Vérifier que le flux n'est plus nécessaire sur ce routeur
            Si I.ElementActif.PimSM.isFluxNécessaire(S, G, RPT, I) = True

                  // Si le flux est encore nécessaire, le Prune ne peut-être pris en compte.
                  Return False

            Fin Si

      Fin Pour tout

      Return True
Fin

Méthode PimSM.isFluxNécessaire ( S, G, RPT,  Interface)
      // S, G et RPT = idem ci-dessus
      // Interface = interface d'entrée du flux traité
Début
      Si S = Null          //Prune (*,G)

            // Si l'interface d'entrée mène au RP du groupe
            // et que la liste des interfaces de sortie pour ce flux(*,G) n'est pas vide
            Si unicast.getRoute ( RP(G) ) = Interface
            Et TIB.Grp(G).ListeInterfacesSortie Not = Liste Vide

                  // Le flux est toujours nécessaire (pour une demande Join ou IGMP)
                  Return True
            Fin Si
      Sinon Si  RPT = False          // Prune (S,G)

            // Si l'interface d'entrée mène à la source
            // et que la liste des interfaces de sortie pour ce flux(S,G) est  vide
            // ou que les entrées ne concernent que des PruneRPT
            Si unicast.getRoute ( S ) = Interface
            Et ( Il existe I
                  tel que I appartient à TIB.Grp(G).Source(S).ListeInterfacesSortie,
                  Et (I.JoinSG = True ou I.IgmpSG = True ) )

                  // Le flux est toujours nécessaire(pour une demande Join ou IGMP)
                  Return True
            Fin Si

      Sinon       // Prune (S,G,RPT)

            // Si l'interface d'entrée mène au RP du groupe
            // Et que toutes les interfaces ayant reçu un Join(*,G) ou un IGMP(*,G)
            //     aillent également reçu un prune (S,G,Rpt)
            Si unicast.getRoute ( RP(G) ) = Interface
            Et ( Pour Tout I dans TIB.Grp(G).ListeInterfacesSortie ,
                        ( I dans TIB.Grp(G).Source(S).ListeInterfacesSortie
                          Et I.PruneRptSG = True ) )

                  // Le flux est toujours nécessaire (pour une demande Join ou IGMP)
                  Return True
            Fin Si

      Fin Sinon

      Return False
Fin
```

6.6.3. La réception d'un message Prune

Les messages Prune sont de deux types :

- les messages Prune (*,G)
- les messages Prune (S,G)
- les messages Prune (S,G,Rpt)

A l'arrivée d'un de ces messages dans le routeur, celui-ci examine son type et le dispatche à l'une des deux méthodes spécialisées (méthodes « PruneG », « PruneSG » et « PruneSGRpt ») de son instance de PimSM.

```
Méthode PruneG ( interfaceEntrée, messagePruneG)
        // interfaceEntrée  = interface d'entrée du message
        // messagePruneG = message Prune (*,G) traité
Début
        NeedPrune = False

        // Si cette demande prune (*,G) peut-être effectuée
        Si isPruneOk (null, G, False, interfaceEntrée)

                // Si l'interface d'entrée est référencée dans le groupe G de la TIB
                S'il existe I dans TIB.Grp(G).ListeInterfacesSortie
                Tel que I = interfaceEntrée

                        // Désactiver le JoinG de cette entrée
                        Si I.Igmp = True
                                I.Join = False
                        Sinon
                                TIB.Grp(G).ListeInterfacesSortie -= I
                        Fin Sinon
                Fin Si

                // Après la prise en compte du Prune,
                // s'il n'existe plus de Join Ou de IGMP pour ce flux
                Si TIB.Grp(G).ListeInterfacesSortie = null

                        // Un Prune sera nécessaire
                        NeedPrune = True

                        // Suppression des hypothétiques Prune(S,G,Rpt)
                        Pour tout S dans TIB.Grp(G).ListeSource
                                Pour tout I dans S.ListeInterfacesSortie
                                        Si I.Igmp = True ou I.Join = True
                                                I.PruneRpt = False
                                        Sinon
                                                S.ListeInterfacesSortie -= I
                                        Fin Sinon
                                Fin Pour tout
                        Fin Pour tout
                Fin Si
        Fin Si

        // Notifier l'arrivée du Join (*,G) via cette interface dans la TIB
        TIB.Grp(G).ListeInterfacesSortie.interface(interfaceEntrée).Join = True

        Si NeedPrune = True

                // Traitement de l'émission des Joins en amont
                sendJoinPrune ( flux(*,G), "Prune")
        Fin Si
Fin


Méthode PruneSG ( interfaceEntrée, messagePruneSG)
        // interfaceEntrée  = interface d'entrée du message
        // messagePruneSG = message Prune (S,G) traité
Début
        NeedPrune = False

        // Si cette demande prune (S,G) peut-être effectuée
        Si isPruneOk (S, G, False, interfaceEntrée)

                // Si l'interface d'entrée est référencée
```

```
                    // dans la source S du groupe G de la TIB
                    S'il existe I dans TIB.Grp(G).Souce(S).ListeInterfacesSortie
                    Tel que I = interfaceEntrée

                            // Désactiver le JoinG de cette entrée
                            Si I.Igmp = True ou I.pruneRpt = True
                                    I.Join = False
                            Sinon
                                    TIB.Grp(G). Souce(S).ListeInterfacesSortie -= I
                            Fin Sinon
                    Fin Si

                    // s'il n'existe plus de Join Ou de IGMP pour ce flux (S,G)
                    Si pour tout I dans TIB.Grp(G).Souce(S).ListeInterfacesSortie,
                      I.Join = False et i.igmp = False

                            // Un Prune sera nécessaire
                            NeedPrune = True
                    Fin Si
            Fin Si

            Si NeedPrune = True

                    // Traitement de l'émission des messages Prune en amont
                    sendJoinPrune ( flux(S,G), "Prune")
            Fin Si
Fin


Méthode PruneSGRpt ( interfaceEntrée, messagePruneSGRpt)
        // interfaceEntrée  = interface d'entrée du message
        // messagePruneSGRpt = message Prune (S,G, Rpt) traité
Début
            NeedPrune = False

            // Si cette demande prune (S,G, Rpt) peut-être effectuée
            Si isPruneOk (S, G, True, interfaceEntrée)

                    // Notifier le prune (S, G, Rpt) dans la TIB
                    TIB.Grp(G).Source(S).Interface(interfaceEntrée).pruneRpt = True

                    // Si toutes les interfaces ayant reçu un Join ont également reçu un PruneRpt
                    Si pour tout I dans TIB.Grp(G).ListeInterfacesSortie, Tel que I.Join = True
                    Il existe Is dans TIB.Grp(G).Source(S).ListeInterfacesSortie,
                    Tel que I = Is Et Is.pruneRpt = True

                            // Un Prune sera nécessaire
                            NeedPrune = True
                    Fin Si
            Fin Si

            Si NeedPrune = True

                    // Traitement de l'émission des messages Prune en amont
                    sendJoinPrune ( flux(S,G), "PruneRpt")
            Fin Si
Fin
```

6.6.4. L'émission de messages Join / Prune

Les algorithmes précédents (de même pour IGMP qui n'a pas été détaillé) permettent de gérer la réception des messages Join et Prune. Ils modifient la TIB en fonction du contenu de ces messages et, en cas de besoin, font appel à la méthode « sendJoinPrune ». Cette méthode est spécialisée pour la gestion de l'émission de ces

messages. Généralement, il est nécessaire d'émettre plusieurs messages différents en réaction à une demande. Ce qui rend cette méthode relativement complexe.

```
Méthode sendJoinPrune ( flux (S,G), type)
        // flux (S,G) = flux concerné par la demande (si S = null => flux (*,G))
        // type  = Type de demande ("Join", "Prune" ou "PruneRpt")
Début
        NeedPrune = False

        //Recherche de la destination du message à expédier
        destination = Null
        Si S = Null ou type = "PruneRpt"          //Join(*,G), Prune(*,G) ou Prune(S,G,Rpt)

                // Ce routeur n'est pas la destination finale
                Si RP(G) Not = This.routeur
                        destination = RP(G)    // = RP du groupe
                Fin Si
        Sinon
                // Ce routeur n'est pas directement connecté à la destination finale
                Si This.routeur.getInterfaceConnectedTo ( S.LAN ) = Null
                        destination = S.LAN    // = Le LAN connecté à la source
                Fin Si
        Fin Sinon

        Si destination Not Null  // La destination n'est pas encore atteinte

                InterfaceSortie = unicast.getRoute (destination )

                Si type = "Join" et S Not Null        // Join(S,G)

                        // Emission d'un Join (S,G) en direction de la source
                        Send New JoinSG (S, G) Via InterfaceSortie

                        // Si besoin, émission d'un Prune (S, G, RPT)
                        Si TIB.Grp(G).ListeInterfacesSortie Not Null
                                sendJoinPrune ( flux(S,G), "PruneRpt")
                        Fin Si

                Sinon Si type = "Join" et S = Null     // Join(*,G)

                        // Emission d'un Join (S,G) en direction du RP
                        Send New Join (*, G) Via InterfaceSortie

                        // Si besoin, émission de Prune (S, G, RPT)
                        // pour chaque source déjà réceptionnée via le SP-tree
                        Pour tout Src dans TIB.Grp(G).ListeSource

                                S'il Existe I dans Src.ListeInterfacesSortie
                                Tel Que I.Join = True ou I.Igmp = True
                                        sendJoinPrune ( flux(Src,G), "PruneRpt")
                                Fin S'il Existe
                        Fin Pour tout

                Sinon Si type = "Prune" et S Not Null          // Prune(S,G)

                        // Emission d'un Prune (S,G) en direction de la source
                        Send New PruneSG (S, G) Via InterfaceSortie

                        // Si besoin, suppression de Prune (S, G, RPT) précédemment émis
                        // ( Si un Join (*,G) est actif un prune(S,G,Rpt) à été émis)
                        Si TIB.Grp(G).ListeInterfacesSortie Not Null

                                // Emission d'un Join (*,G)
                                // ( Celui-ci sera suivi des Prune (S, G, Rpt) nécessaires )
                                sendJoinPrune ( flux(*,G), "Join")
                        Fin Si

                Sinon Si type = " Prune " et S = Null          // Prune (*,G)
```

```
                    // Emission d'un Prune (*,G) en direction de la source
                    Send New PruneSG (*, G) Via InterfaceSortie

                Sinon   // Prune(S,G, Rpt)

                    // Emission d'un Prune (S,G,Rpt) en direction de la source
                    Send New PruneSGRpt (S, G) Via InterfaceSortie

                Fin Sinon
            Fin Si
Fin
```

6.6.5. La réception d'un message multicast en transit

A ce stade, les routeurs Pim SM sont capables de gérer la diffusion à travers le domaine de la structure des arbres et de gérer les informations contenues dans leur TIB. Il ne leur reste plus qu'à être capable de recevoir un message multicast (méthode « transit ») et, en fonction du contenu de sa TIB, le réexpédier sur les interfaces nécessaires. Ces interfaces sont connues grâce à la méthode « getListeInterfaceSortie » que nous verrons au point 6.6.6.

De plus, à la réception d'un message multicast, il est nécessaire de vérifier que ce routeur n'est pas le DR de la source du message et, dans l'affirmative, initier la gestion de ce besoin (voir point 6.6.7.).

```
Méthode transit ( interfaceEntrée, messageMulticast)
        // interfaceEntrée  = interface d'entrée du message
        // messageMulticast = message multicast  traité
Début
            //Obtention de la liste des interfaces de sortie pour ce flux
            lstInterf = getListeInterfaceSortie (messageMulticast.Source,
                                                 messageMulticast.Groupe,
                                                 interfaceEntrée )

            // réexpédier le message
            Pour tout I dans _LstInterf
            Tel que I Not = interfaceEntrée
                    Send messageMulticast Via I
            Fin pour tout

            // Si ce routeur est le DR de la source et que ce flux (S,G) n'est pas encore enregistré
            Si DR( messageMulticast.Source ) = This.routeur
            Et TIB.Groupe(G).Source(S).registerDone Not = True
                    // initié la gestion du mode register
                    sendRegister (messageMulticast)
            Fin Si
Fin
```

6.6.6. Création de la liste des interfaces de sortie

La méthode transit vue ci-dessus fait appel à la méthode « getListeInterfaceSortie » et ce, afin de créer la liste des interfaces par lesquelles un message arrivé via l'interface « interfaceEntrée » doit-être dispatché. Il est à remarqué que cette liste d'interface peut contenir l'interface d'entrée du message.

```
Méthode getListeInterfaceSortie (source, groupe, interfaceEntrée )
        // source = source du flux
        // groupe = groupe du flux
        // interfaceEntrée  = interface d'entrée du message
```

```
Début
        // Initialiser la liste des interfaces
        lstInterface = Null

        // Si la TIB contient des informations pour le groupe demandé
        Si TIB.Groupe(G) Not Null

                // Si l'interface d'entrée mène à la source
                Si unicast.getRoute (source ) = interfaceEntrée

                        lstInterface = OIListSG (source, groupe)

                // Sinon, si elle mène au RP du groupe
                Sinon Si unicast.getRoute ( RP(groupe) ) = interfaceEntrée

                        lstInterface = OIListG (source, groupe)

                Fin Sinon
        Fin Si

        Retour lstInterface

Fin
```

Cette méthode vérifie en premier si l'interface d'entrée mène à la source et, dans l'affirmative, obtient la liste des interfaces à partir de la méthode « OIListSG ». Celle-ci cumule les interfaces par lesquelles le SP-Tree et le RP-Tree doit-être propagé. Par contre, si l'interface ne mène pas à la source mais au RP du groupe, créé la liste des interfaces uniquement pour le RP-Tree à l'aide de la méthode « OIListG ».

```
Méthode OIListSG (source, groupe)
Début
        // Commencer par récupérer les interfaces pour le RP-Tree
        lstInterface = OIListG (source, groupe)

        Pour tout I dans TIB.Grp(groupe).Souce(source).ListeInterfacesSortie
        Tel que  I.Join = True ou I.Igmp = True

                Si I Not In lstInterface
                        lstInterface + = I
                Fin si

        Fin Pour tout

        Retour lstInterface
Fin


Méthode OIListG (source, groupe)
Début
        LstInterface = Null

        Pour tout I dans TIB.Grp(groupe).ListeInterfacesSortie

                // Si le routeur est DR du LAN connecté à cette interface
                // et qu'un Query IGMP (*,G) y est actif
                Si I.Igmp = True

                        lstInterface + = I

                // sinon, si un Join (*,G) est arrivé via cette interface
                // et qu'aucun Prune (S,G,Rpt) n'est arrivé par cette même interface
                Sinon Si I.Join = True
                Et S'il existe Is dans TIB.Grp(groupe).Source(source)ListeInterfacesSortie
                   Tel que Is.pruneRpt = False

                        lstInterface + = I
```

```
                Fin si

            Fin Pour tout

            Retour lstInterface
Fin
```

6.6.7. La gestion des messages Register et RegisterStop

A la fin de la méthode transit, celle-ci vérifie si le routeur est le DR du LAN sur lequel la source du message est connectée. Si c'est le cas, elle vérifie l'état du mode Register du flux (S,G) et si besoin, initie l'expédition d'un message Register vers le RP via l'intermédiaire de la méthode « sendRegister ».

```
Méthode sendRegister (messageMulticast)
Début
        // Obtention du groupe multicast destination du message
        grp = messageMulticast.Groupe

        // Obtention de la source du message
        src = messageMulticast.Source

        // Si l'enregistrement n'est pas encore terminé
        Si TIB.Groupe(grp).Source(src).RegisterDone Not True

                // Si ce routeur est le RP, pas besoin de Register
                Si this.routeur = RP(Grp)
                        TIB.Groupe(grp).Source(src).RegisterDone = True
                Sinon
                        // émettre un message Register de ce routeur vers le RP
                        // le message Register encapsule le message multicast source
                        msgRegister =New Register(this.routeur, RP(Grp), messageMulticast)
                        send msgRegister via unicast.getRoute( Rp(grp) )
                Fin Sinon
        Fin Si
Fin
```

Le message Register est transféré de la sorte de routeur en routeur. Lorsque l'un de ceux-ci le réceptionne, il est transmis à la méthode « RegisterIn » de PimSm. Son rôle est d'analyser le message, de le traiter si le routeur est le RP du groupe ou de le transférer au routeur suivant en direction du RP.

```
Méthode registerIn (msgRegister)
Début
        // Obtention du groupe multicast destination du message
        grp = msgRegister.Groupe

        // Obtention de la source du message
        src = msgRegister.Source

        // Si le message est à destination
        Si RP(grp) = this.routeur

                // récupérer le message multicast initial
                messageMulticast = msgRegister.Msgmulticast

                //Obtention de la liste des interfaces de sortie pour ce flux
                lstInterf = getListeInterfaceSortie (messageMulticast.Source,
                                                     messageMulticast.Groupe,
                                                     interfaceEntrée )

                // réexpédier le message
                Pour tout I dans LstInterf
                Tel que I Not = interfaceEntrée
```

```
                    Send messageMulticast Via I
                Fin pour tout

                // expédier un message Register Stop en direction du DR
                send New msgRegisterStop (src, grp) Via unicast.getRoute( src )

        Sinon
                // émettre un message Register de ce routeur vers le RP
                send msgRegister via unicast.getRoute( Rp(grp) )
        Fin Sinon
Fin
```

Lors du traitement du message, si la liste des interfaces de sortie n'est pas vide, le RP commence par expédier en multicast le message multicast contenu dans le Register. A la suite de cela il émet un Register Stop en direction du DR à l'origine du Register. Ce message est transféré de la sorte de routeur en routeur et, à la réception de celui-ci, traité par la méthode « registerStopIn » de PimSm.

```
Méthode registerStopIn (msgRegisterStop)
Début

        // Si le message est à destination
        Si RP(grp) = this.routeur

                // Terminer la phase d'enregistrement du flux
                grp = msgRegisterStop.Groupe
                src = msgRegisterStop.SourceFlux
                TIB.Groupe(grp).Source(src).RegisterDone = True

        Sinon
                // émettre un message Register de ce routeur vers le RP
                send msgRegister via unicast.getRoute( Rp(grp) )
        Fin Sinon
Fin
```

7. Conclusion

Dans ce travail, nous avons commencé par situer la problématique des communications multicast dans le monde de l'informatique. Après avoir expliqué l'apparition de ce besoin sur Internet lié à son expansion, nous avons comparé les différents modes d'émissions d'information disponibles sur ce média.

Internet est régit par de nombreux protocoles et règles de communication. Sur cet ensemble, vient se greffer le protocole de gestion du trafic multicast sur le backbone. Celui-ci doit donc pouvoir interagir avec un sous-ensemble de protocole existant qu'il est important de connaître. A la fin de la première partie du deuxième chapitre, le lecteur est suffisamment familiarisé avec ces règles de bases.

Dans la suite de ce deuxième chapitre, nous avons réalisé une approche théorique des différentes méthodes permettant de diffuser les flux multicast sur le backbone. Chaque technique ayant ses avantages et ses inconvénients, la mise en évidence de ceux-ci à permis de mieux cerner les besoins propres à la réalisation de cette tâche sur un réseau aussi vaste qu'Internet. Cette étude théorique à été complétée par une rapide présentation de deux implémentations fonctionnelles (DVMRP et PIM DM). Celles-ci restant plus adaptées à un domaine tel que l'Intranet d'une société ou d'une université.

Cette présentation nous a permis de passer à l'étude approfondie du protocole PIM SM, au cours de laquelle nous avons réalisé :

- l'étude des spécificités (adaptée aux besoins d'Internet) de ce protocole
- La présentation des différents éléments constitutifs d'un domaine PIM SM et de leurs rôles.
- L'étude des différentes phases du fonctionnement du protocole.

Cette étude nous à permis de nous attaquer à la réalisation de l'objectif principal de ce travail. C'est à dire la réalisation d'un simulateur PIM SM. Le quatrième chapitre nous à permis de définir le type de simulateur envisagé et de réaliser une rapide spécification des besoins des futurs utilisateurs de celui-ci.

Sur base des objectifs fixés, dans le cinquième chapitre, nous avons réalisé l'analyse des objets permettant la simulation d'un domaine multicast, de leurs rôles respectifs et des interactions existantes entre ceux-ci. Nous nous sommes ensuite attaqué à la conception de l'interface. Le domaine et l'interface étant indépendants l'un de l'autre, nous avons complété l'ensemble d'objets chargés de réaliser la connexion entre le domaine simulé et l'interface utilisateur. C'est à dire transférer aux objets du domaine les demandes de l'utilisateur et retourner à l'interface les modifications d'état des objets du domaine et des différents évènements se déroulant sur celui-ci.

Dans le sixième chapitre, nous nous sommes penchés sur l'implémentation du domaine. Etant donné qu'il s'agit d'une simulation et non d'une implémentation réelle des différents protocoles nécessaires, un bon nombre de choix ont du être réalisés et ce, afin de permettre une implémentation réaliste (vis à vis de l'utilisateur), réalisable (par un développeur en un temps relativement court) et fonctionnel. Une implémentation suffisamment fonctionnelle ayant été réalisée, nous pouvons en conclure que ces objectifs ont étés atteint. Il serait cependant très intéressant de poursuivre le développement du simulateur afin d'arriver à une version finale stable et complète.

8. Annexes

8.1. Références bibliographiques

Andrew S. Tanenbaum, *Computer Networks (third Edition)*, Prentice Hall, 1996

Thomas A. Maufer, *IP Fundamentals*, Prentice Hall, 1999

Huitena Christian, *Routing in the Internet*, Englewood Cliffs, 1995

I. Rudenko, *Configuration IP des Routeurs Cisco*, Eyrolles, 2000

McGraw, *CCNP Routing Study Guide,* Syngress Media

Références Internet intéressantes :

Le sujet traité dans ce travail étant extrêmement lié à Internet, il est normal d'y retrouver un très grand nombre de documents de toutes origines traitant de celui-ci. Vous trouverez ci-dessous quelques URL intéressantes et disponibles au moment de la rédaction de ce mémoire.

http://www.ietf.org/internet-drafts/draft-ietf-pim-sm-v2-new-02.txt

http://www.microsoft.com/technet/win2000/pimsm2.asp

http://www.stardust.com/multicast/whitepapers/glossary.htm

http://www.cisco.com/

8.2. Les différents timers de PIM SM

Dénomination	Valeur par défaut	Description
Bootstrap Period	60 secondes	Période entre deux émissions d'un message "Bootstrap" par le BSR du domaine.
Bootstrap Timeout	2 * Bootstrap Period + 10 secondes	Période après laquelle l'éléction d'un BSR est débutée si aucun message "Bootstrap" n'est réceptionné par les CBSR en provenance du BSR actuel.
Hello Period	30 secondes	Période séparant l'émission de deux messages "HELLO" sur chaque interface active d'un routeur.
Hello Holdtime	3.5 * Hello Period	Période pendant laquelle un message "Hello" reste valable après sa réception. (sa valeur est en réalité définie dans le message "Hello" et non dans le routeur qui réceptionne celui-ci)
Join-Prune Holdtime	Définie dans le message "Join-Prune"	Période pendant laquelle un message "Join-Prune" reste valable après sa réception (elle est définie dans chaque message "Join-Prune")
Join-Prune Override Interval	5 secondes	Courte période pendant laquelle un routeur à l'occasion de contrer un message "Prune" émis sur un LAN
Join-Prune Periodic	60 secondes	Période séparant l'émission de deux messages "Join-Prune"
Assert Time	180 secondes	Période pendant laquelle un message "Assert" reste valable.
Assert Override Interval	5 secondes	Petite période précédant l'expiration d'un timer "Assert Time" durant laquelle un routeur responsable de l'émission d'un flux sur un LAN retransmet un message "Assert" pour ce flux.
Keepalive Period	210 secondes	Période pendant laquelle un flux reste actif après la réception d'un paquet appartenant à ce flux même en l'absence d'un "Join" pour ce groupe
RP Keepalive Period	95 secondes	Période pendant laquelle un RP continue à gérer un flux après l'émission d'un message "Register Stop"
C-RP Timeout	Définie dans le message	Période pendant laquelle un message "Candidate-RP" reste valable
C-RP Advertisement Period	60 secondes	Période séparant l'émission de deux messages "Candidate-RP"

Period		"Candidate-RP"
Dénomination	**Valeur par défaut**	**Description**
Register Suppression Time	60 secondes	Période pendant laquelle un DR arrête l'émission d'un flux via encapsulation dans un mesage "Register" suite à la réception d'un message "Register Stop"
Register Probe Time	5 secondes	Courte période précédant l'expiration d'un timer "Register Suppression Time" pendant laquelle un DR à l'occasion d'émettre un message "Null-register" à destination du RP (ce message lui donnant la possibilité de réexpédier directement un nouveau message "Register Stop")

8.3. Le mode d'emploi du simulateur

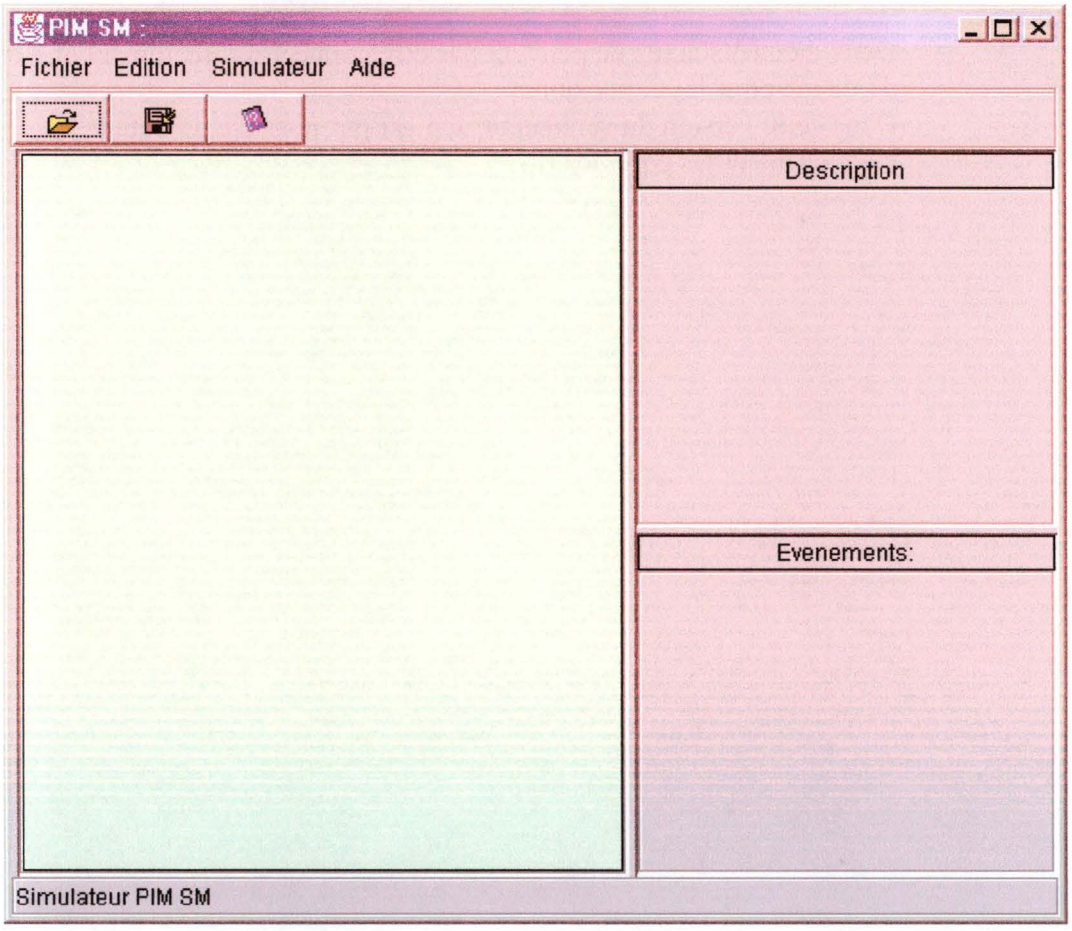
Mon intention lors de la rédaction de ces quelques lignes n'est pas de fournir un manuel complet et exhaustif. Mais, simplement de fournir les informations permettant l'exécution et l'utilisation de celui-ci. Aucunes explications du fonctionnement du protocole Pim SM ne seront fournies ici.

1. Exécution de l'application

Le fichier PimSM.jar renferme toutes les classes et autres éléments nécessaires à l'exécution du simulateur. Il suffit d'exécuter en ligne de commande la commande suivante (attention à la caste) : java –jar PimSM.jar. Ce qui active l'interface graphique

Remarque : Le simulateur nécessite une interface graphique (Win32 ou Xwindow). Il à été développé sous Java 1.3. Il est donc nécessaire de posséder sur la station d'une version du JDK 1.3 (Attention : la variable d'environnement Path doit pointer sur la directory bin du jdk)

Une version du JDK 1.3 est disponible à : http://www.sun.com/software/download



2. Eléments disponibles

Lors du lancement de l'application, le domaine simulé est directement actif.
Cependant, initialement, celui-ci ne comprend aucun élément. Il sera nécessaire de les ajouter, d'établir les connexions entre ceux-ci et de définir les propriétés de chacun d'entre eux.

Les éléments disponibles sont de deux types :

1. Les éléments actifs : Ceux-ci participent de manière dynamique aux flux d'information. Ils émettent, gèrent et reçoivent les différents flux.
    a. Les stations : Ils agissent en temps qu'émetteur de trafic et/ou de receveur. Dans ce simulateur, elles ne peuvent pas contenir plus d'une interface.
    b. Les routeurs : Leur tâche principale est la gestion du trafic multicast et, par définition, ne sont pas limités à une seule interface.

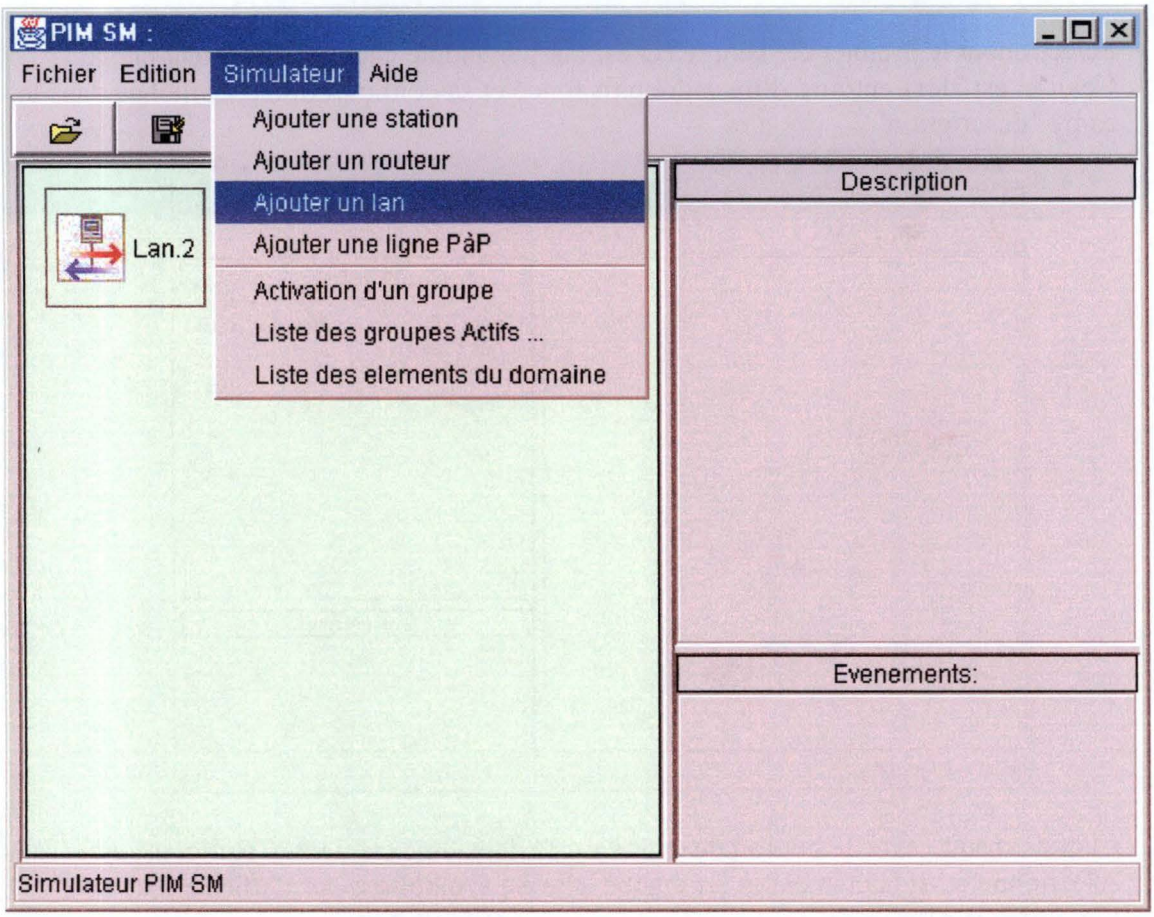
2. Les connexions : Elles permettent de relier deux éléments actifs entre eux. De plus, ces objets sont responsables de la gestion des adresses IP. Chaque connexion s'attribue automatiquement une adresse réseau. A chaque établissement d'une liaison avec un élément actif, la connexion attribue à l'interface de celui-ci une adresse IP unique. Elles sont de deux types :
    a. Liaison Point à Point : Elles permettent uniquement la connexion de deux routeurs.
    b. Lan : Ils permettent l'interconnexion d'un plus grand nombre d'éléments. Dans ce simulateur, les stations ne peuvent être reliées qu'à un Lan.

Remarques :

- Deux éléments actifs ne peuvent pas être connectés directement et il en est de même pour les connexions.
- Dans cette version du simulateur, il n'est pas possible de définir manuellement les adresses IP.

3. Ajout d'un élément

Avant tout, il est nécessaire d'ajouter au domaine un certain nombre d'élément.
Pour ce faire, il vous suffit d'utiliser le menu Simulateur.

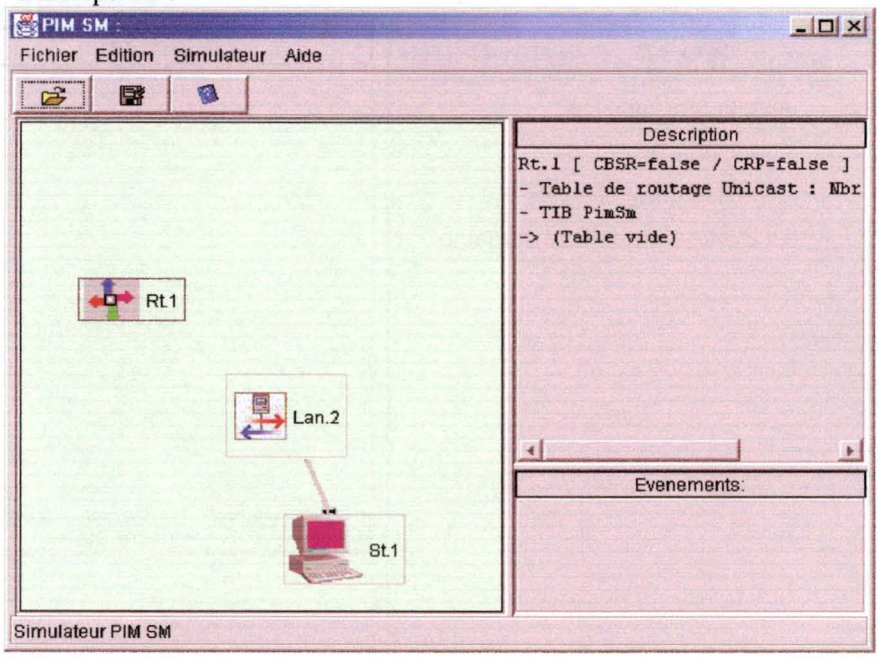


L'élément est automatiquement ajouté en haut à gauche. Il est ensuite possible de placer celui-ci à l'emplacement désiré à l'aide de la souris par simple drag de l'élément.

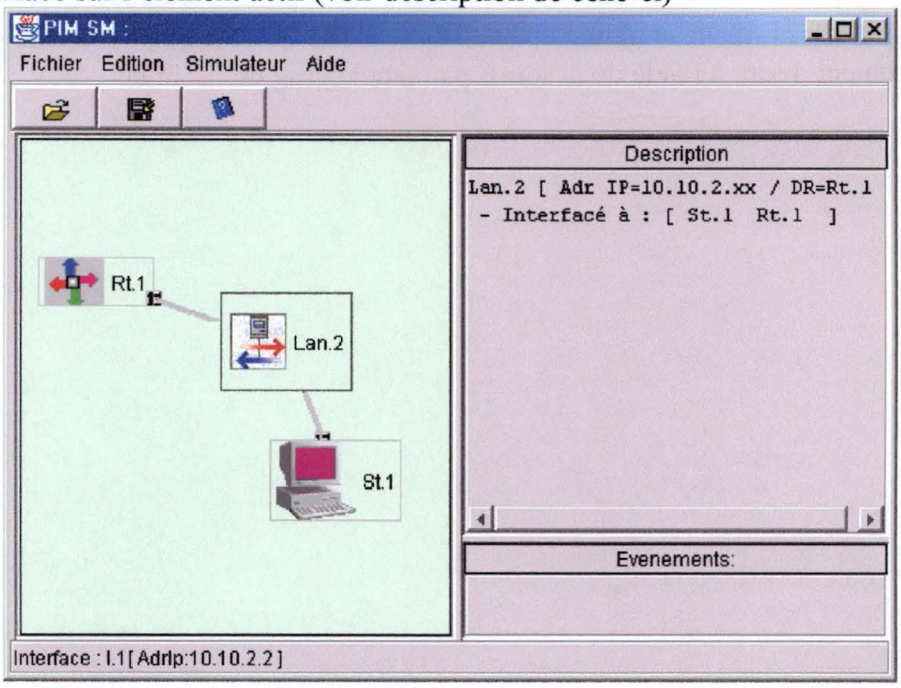
4. Connexion de deux éléments

Les éléments nouvellement ajoutés au domaine ne sont connectés à rien. Il est nécessaire de le faire manuellement. Pour ce, la manière la plus simple est d'utiliser la souris. La marche à suivre est la suivante :

1 Sélectionner le premier élément. Ceci est fait par simple click sur l'élément choisi. Celui-ci est alors entouré d'un cadre plus foncé et sa configuration est affichée dans le cadre 'description'.



2 Cliquer ensuite avec le bouton de droite sur le deuxième élément. Celui-ci est alors sélectionné et, si la connexion est établie, elle est symbolisée par l'affichage de l'interface sur l'élément actif (voir description de celle-ci)

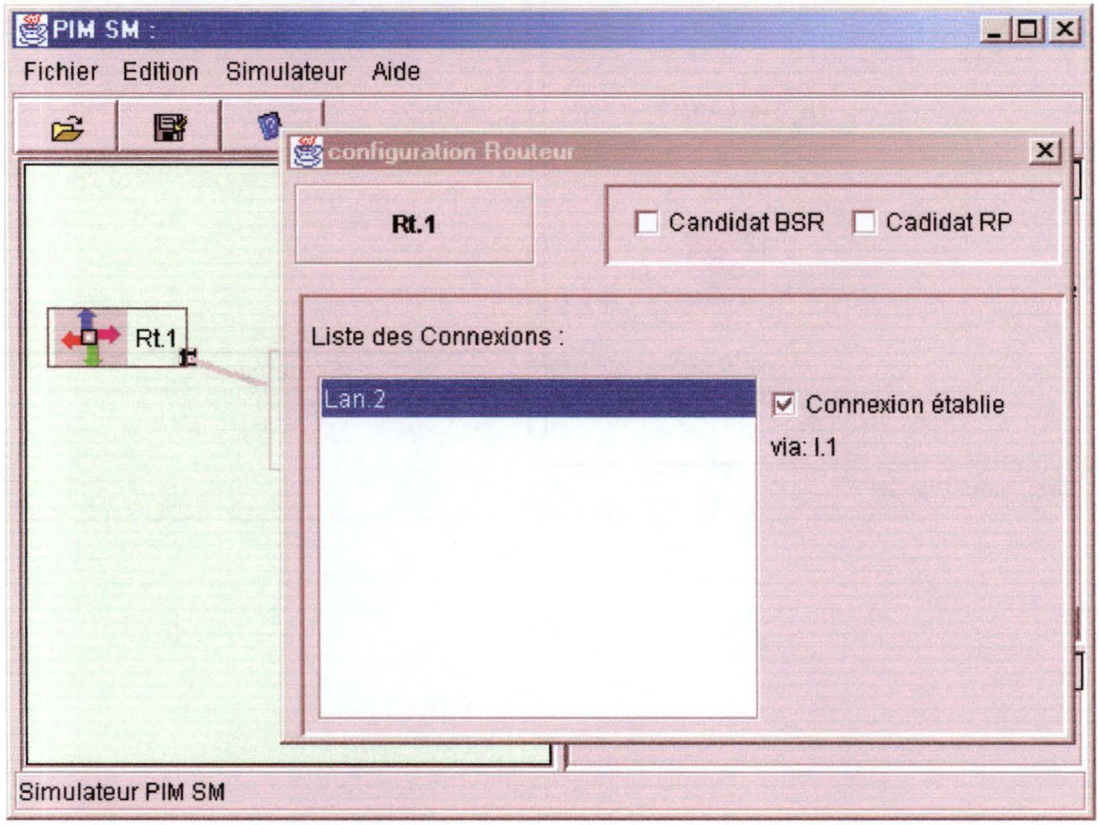


5. Suppression d'une connexion

La suppression d'une connexion n'est pas possible à l'aide de la souris. Pour ce faire, il est nécessaire de passer par la boîte la configuration de l'élément Actif (Station ou Routeur) pour supprimer l'interface. Pour activer cette boîte de dialogue, vous pouvez soit

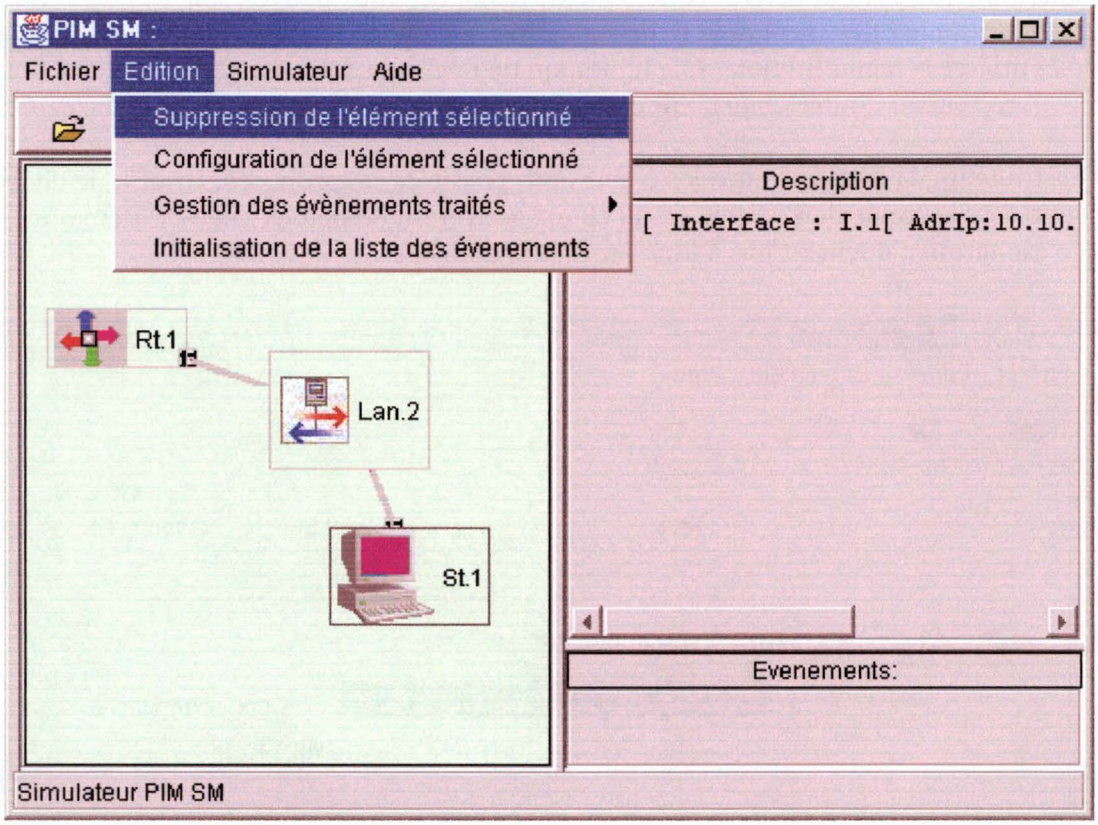
- sélectionner l'élément actif (Clic sur celui-ci)
  et utiliser le menu Edition / Configuration de l'élément sélectionné.
- directement double cliquer sur celui-ci.

Pour les routeurs, la liste de toutes les connexions possibles est proposée. Il suffit de choisir dans cette liste la connexion désirée et, au choix, activer ou désactiver celle-ci. Remarque, le routage est automatiquement mis à jour sur tout le domaine.



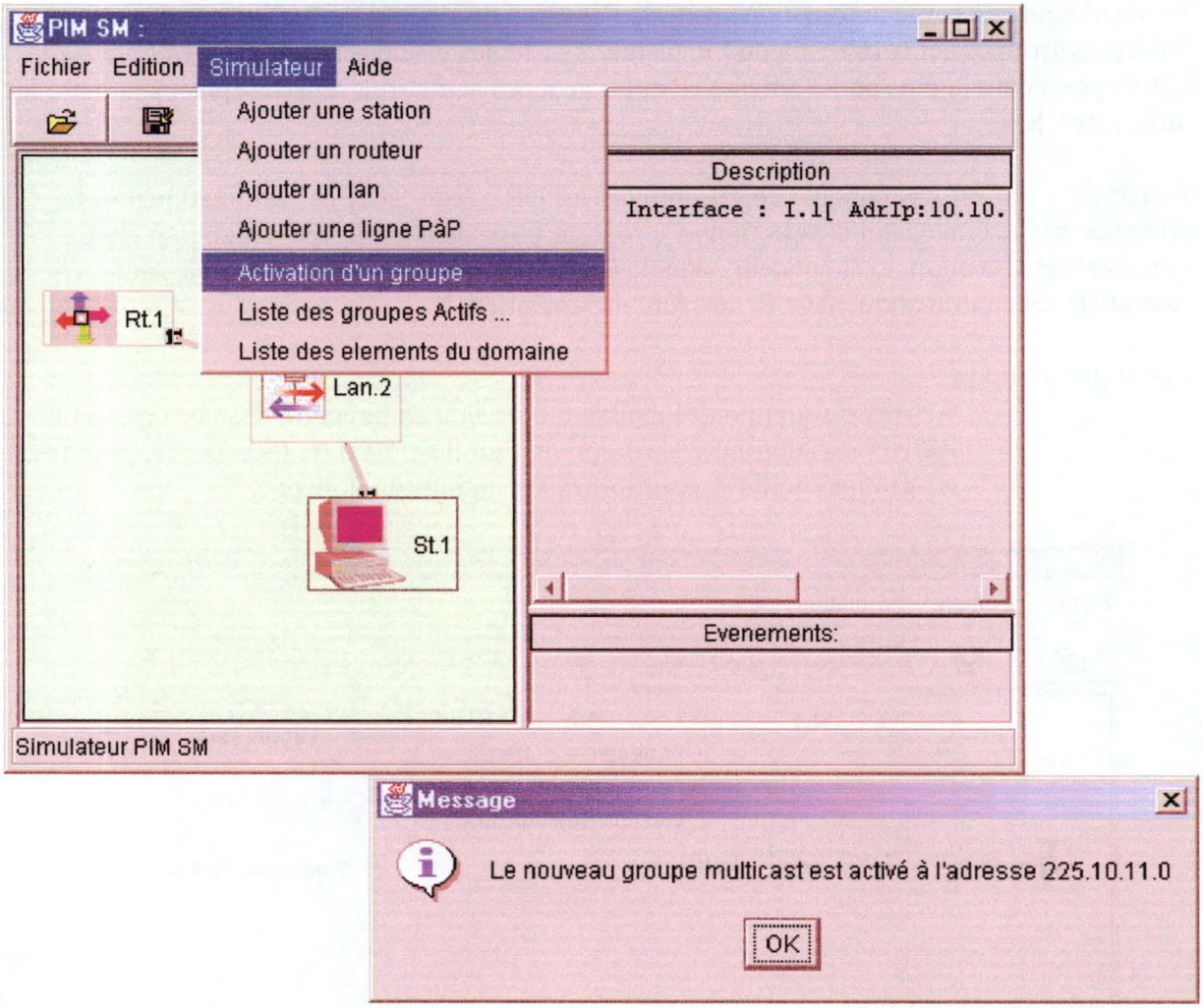
6. Suppression d'un élément

Pour la suppression d'un élément, il suffit de sélectionner l'élément à supprimer et utiliser le menu Edition / Suppression de l'élément sélectionné.

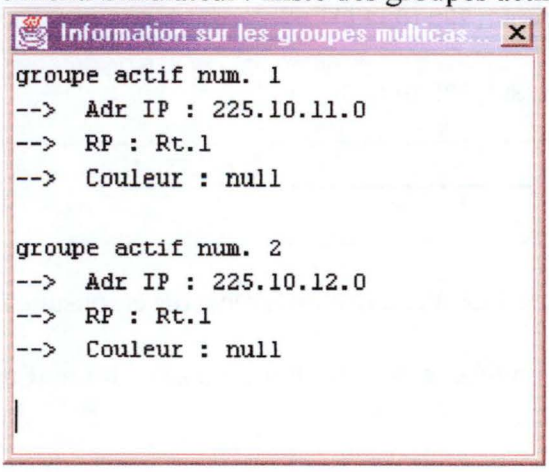


7. Activation d'un groupe

Au démarrage, aucun groupe multicast n'est activé sur le domaine. Pour permettre la simulation du protocole, il est nécessaire de commencer par activer au moins un groupe multicast. Pour ce faire, utilisez le menu Simulateur / Activation d'un groupe. Le simulateur active un groupe en lui fournissant automatiquement une adresse IP multicast.



Vous avez la possibilité d'activer un nombre quelconque de groupe et d'obtenir leur configuration en utilisant le menu Simulateur / Liste des groupes actifs



8. Configuration d'un élément

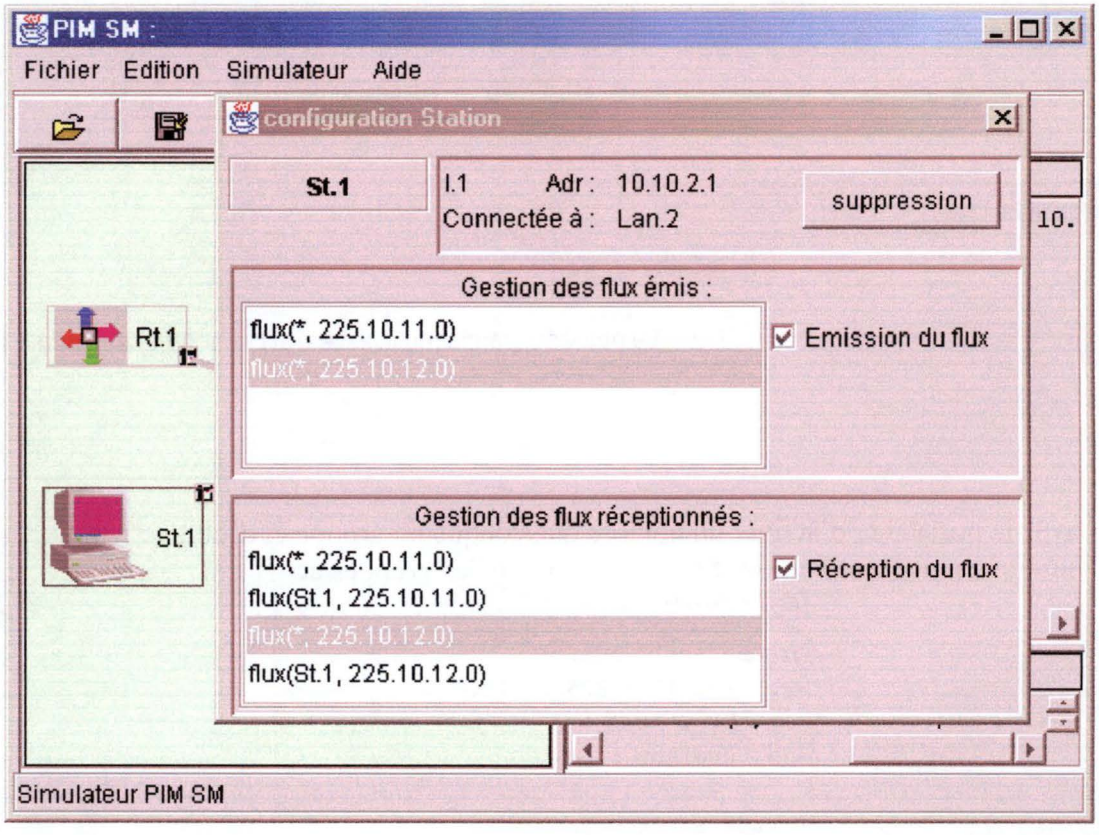
Outre la possibilité de modifier les interfaces des éléments actifs, vous pouvez

- dans le cas d'un routeur :
    - le configurer comme CBSR
    - le configurer comme CRP

Le simulateur se charge automatiquement de l'élection du BSR et de l'affectation des RP aux différents groupes actifs (CF : menu Simulateur / Liste des groupes actifs). Le BSR élu est le CBSR possédant la plus petite adresse IP et les groupes actifs sont répartis également entre les différents CRP.

Remarque : Il n'est pas possible de sélectionner les DR. Cette tâche est automatiquement effectuée par le Lan. Celui-ci sélectionne le routeur possédant la plus petite adresse IP. Ce qui, dans cette version du simulateur, signifie le premier routeur relié à ce Lan (les adresses sont attribuées automatiquement de manière incrémentielle).

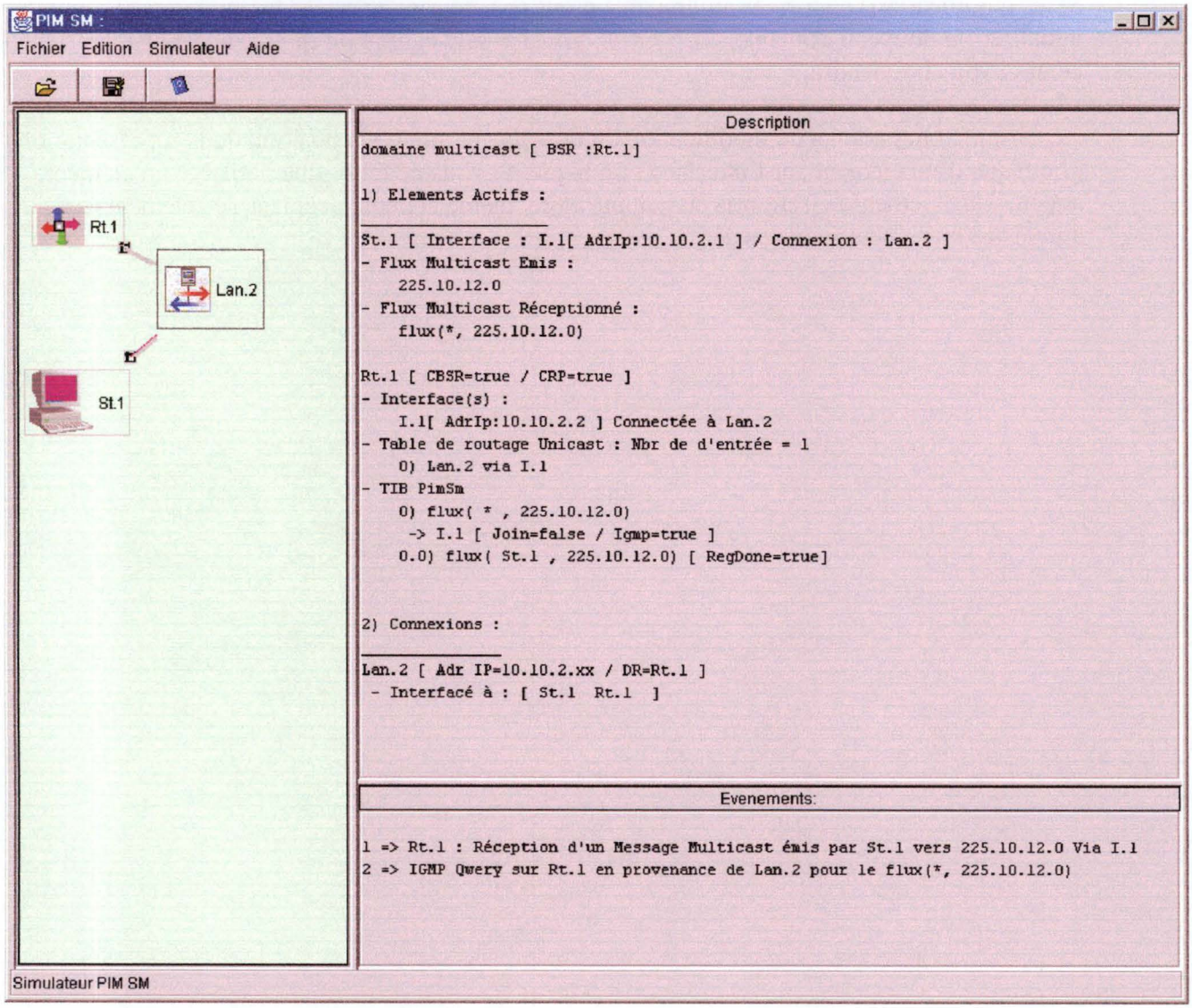
- pour une station :
    - Activer ou supprimer l'émission d'un flux en direction d'un groupe actif
    - Activer ou supprimer la réception d'un flux (*,G) ou (S,G) et ce, pour tous les groupes actifs et pour toutes les stations du domaine.



Pour ce faire, sélectionner le flux à émettre ou à recevoir et, ensuite cocher ou décocher la CheckBox associée.
Remarque : l'activation d'un flux émis se traduit par l'expédition d'un message

9. La gestion des évènements

Les modifications demandées sont automatiquement réalisées sur le domaine. Les évènements traités sont automatiquement listés et leur influence sur les éléments du domaine sont visible sur la description de ceux-ci. De plus, en cliquant sur la représentation graphique du domaine (Pas sur un élément), la description de tout le domaine est affichée dans la description. Il est possible d'obtenir cette même description dans une fenêtre indépendante via le menu Simulateur / Liste des éléments du domaine. Ceci permettant de comparer les modifications apportées aux éléments du domaine suite à une quelconque action.



Les zones de texte sont directement éditables. Ils est donc possible de les modifier, d'ajouter des commentaires et de copier le contenu (ou une partie) dans le presse papier.

Pour simplifier la lecture des événements, il est toujours possible de réinitialiser la liste en utilisant le menu Edition / Initialisation de la liste des évènements.

10. Gestion de fichier

A tout moment, il est possible

- de sauvegarder le domaine sous son nom courant ou sur un nouveau nom
- d'initialiser un nouveau domaine
- de recharger un domaine précédemment sauvegarder.

Ceci est réalisé à l'aide des différentes options du menu fichier.

Si les modifications apportées au domaine n'ont pas été sauvegardées lorsque vous désirez initialiser un nouveau domaine, charger un domaine sauvegardé ou quitter le simulateur, celui-ci vous préviendra.

Remarque : On parle ici de modification du domaine et non de la position de la représentation graphique d'un élément sur l'interface graphique. Si vous ne faites que déplacer un élément, aucun avertissement ne sera pris en compte et ce, même si l'emplacement des éléments est enregistré lors d'une sauvegarde.

Remarques Générales

Cette version du simulateur bien que fonctionnelle, n'est pas optimisée du point de vue interface utilisateur. Il s'agit ici d'une version béta

- La barre de tâche bien que présente, n'est pas implémentée.
- La suppression d'un groupe actif n'est pas impléméntée.
- Le choix par l'utilisateur des évènements pris en compte n'est pas implémenté.
- Il n'est pas possible de demander une gestion des évènements en mode pas à pas
- Il n'est pas possible de sauvegarder dans des fichiers log le contenu des fenêtres texte.
- Les fenêtres d'ouverture et sauvegarde de fichier ne prennent pas automatiquement en compte un type de fichier particulier « *.pim » et ne s'ouvrent pas automatiquement dans la directory du simulateur.
- L'aide n'est pas implémentée
- Les menus contextuels ne sont pas implémentés
- ...

8.4. Le code du simulateur

8.4.1. Interface utilisateur

(Package simulateurpimsm)

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author Dupuis Eric
 * @version 1.0
 */

package simulateurpimsm;

import javax.swing.UIManager;
import java.awt.*;


/**
 * SimPimSM :
 * ----------
 *
 * Lancement de l'application
 * (Création et affichage de la frame principale)
 */

public class SimPimSM
{
  boolean packFrame = false ;

  /**
   * Construction de l'application
   */
  public SimPimSM()
  {
    Frame1 frame = new Frame1();
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame)
    {
      frame.pack();
    }
    else
    {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height)
    {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width)
    {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true );
  }

  /**
   * Méthode principale
   */
  public static void main(String[] args)
  {
    try
    {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    new SimPimSM();
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm;

import  java.awt.*;
import  java.awt.event.*;
import  javax.swing.*;
import  javax.swing.border.*;
import  com.borland.jbcl.layout.*;

import  java.io.*;
import  java.util.Vector;

import  simulateurpimsm.domaine.*;
import  javax.swing.event.*;

/**
 * Frame1 :
 *
 * Fenêtre principale
 * (Gestion de toutes les sous-frames et du menu
 *  + initialisatation et gestion des interactions avec le domaine)
 */
public  class  Frame1 extends  JFrame
{
  // Domaine Multicast
  Domaine domaine = null ;

  // Représentation graphique des objets du domaine
  Vector domaineGph = new Vector();

  // objet du domaine sélectionné
  static   ObjGraphique objSelectionne = null ;

  // Instance de 'ObjGraphique' permettant l'initialisation du système
  ObjGraphique objGraphiqueConfig = null ;

  // Instance de 'ObjInterfaceGraphique' permettant l'initialisation du système
  ObjInterfaceGraphique objInterfaceGraphiqueConfig = null ;

  // Instance de 'GestionEvenements' permettant l'initialisation du système
  GestionEvenements gestionEvenement = null ;

  // Etat de l'interface
  boolean  simulationActive = false  ;

  // Elements utilisés pour l'interface
  String frameTitle = new String ("PIM SM : ");
  String nomFichier = null ;
  boolean  dirty = false  ; // le domaine actif a-t-il été modifié depuis l'ouverture du fichier?
  String enteteFichier = new String("Simulateur PIM SM vesrsion 1.00 (Dupuis Eric)");

  // Elements de l'interface:
  JPanel contentPane;
  JMenuBar jMenuBar1 = new JMenuBar();
  JMenu jMenuFile = new JMenu();
  JMenuItem jMenuFileExit = new JMenuItem();
  JMenu jMenuHelp = new JMenu();
  JMenuItem jMenuHelpAbout = new JMenuItem();
  JToolBar jToolBar = new JToolBar();
  ImageIcon image1;
  ImageIcon image2;
  ImageIcon image3;
  ImageIcon imgStation;
  ImageIcon imgRouteur;
  ImageIcon imgLan;
  ImageIcon imgPaP;
  ImageIcon imgInterface;
  JLabel statusBar = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();
  JMenuItem jMenuItemNew = new JMenuItem();
  JMenuItem jMenuItemOpen = new JMenuItem();
  JMenuItem jMenuItemSave = new JMenuItem();
  JMenuItem jMenuItemSaveAs = new JMenuItem();
  JMenu jMenuSimul = new JMenu();
  JMenuItem jMenuItemHelp = new JMenuItem();
  JMenuItem jMenuItemGstGroupesActifs = new JMenuItem();
  JButton jButton5 = new JButton();
  JButton jButton6 = new JButton();
  JButton jButton10 = new JButton();
  JLabel jLabelStatus = new JLabel();
  TitledBorder titledBorder1;
  JSplitPane jSplitPane1 = new JSplitPane();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JLabel jLabelEvenement = new JLabel();
  TitledBorder titledBorder2;
  JTextArea ListeEvenements = new JTextArea();
  JScrollPane jScrollPane2 = new JScrollPane();
  BorderLayout borderLayout3 = new BorderLayout();
  JPanel DomaineGraphique = new JPanel();
  XYLayout xYLayout1 = new XYLayout();
  JMenuItem jMenuItemAddStation = new JMenuItem();
  JMenuItem jMenuItemAddLan = new JMenuItem();
  JMenuItem jMenuItemAddRouteur = new JMenuItem();
  JMenu jMenuEdit = new JMenu();
  JMenuItem jMenuItemSupElement = new JMenuItem();
  JMenuItem jMenuItemConfigElement = new JMenuItem();
  JMenu jMenu1 = new JMenu();
  JCheckBoxMenuItem jCheckBoxMenuItem1 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem2 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem3 = new JCheckBoxMenuItem();
  JMenuItem jMenuItem7 = new JMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem4 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem5 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem6 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem7 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem8 = new JCheckBoxMenuItem();
  JCheckBoxMenuItem jCheckBoxMenuItem9 = new JCheckBoxMenuItem();
  JMenuItem jMenuItem1 = new JMenuItem();
  JMenuItem jMenuItemAddPaP = new JMenuItem();
  JFileChooser jFileChooser1 = new JFileChooser();
  JLabel jLabelDescription = new JLabel();
  JTextArea description = new JTextArea();
  BorderLayout borderLayout2 = new BorderLayout();
  JSplitPane jSplitPane2 = new JSplitPane();
  JPanel jPanel3 = new JPanel();
  JPanel jPanel4 = new JPanel();
  BorderLayout borderLayout4 = new BorderLayout();
  BorderLayout borderLayout5 = new BorderLayout();
  JScrollPane jScrollPane1 = new JScrollPane();
  JScrollPane jScrollPane3 = new JScrollPane();
  JMenuItem jMenuItemActivationGroupe = new JMenuItem();
  JMenuItem jMenuItem2 = new JMenuItem();

  /**Construct the frame*/
  public  Frame1()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
```

```
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  /**
   * initialisation
   */
  private void jbInit() throws Exception
  {
    imgStation   = new ImageIcon(simulateurpimsm.Frame1.class .getResource("imgStation.gif"));
    imgRouteur   = new ImageIcon(simulateurpimsm.Frame1.class .getResource("imgRouteur.gif"));
    imgLan       = new ImageIcon(simulateurpimsm.Frame1.class .getResource("imgLan.gif"));
    imgPaP       = new ImageIcon(simulateurpimsm.Frame1.class .getResource("imgPaP.gif"));
    imgInterface = new ImageIcon(simulateurpimsm.Frame1.class .getResource("imgInterface.gif"));

    this .domaineGph = new Vector();

    image1 = new ImageIcon(simulateurpimsm.Frame1.class .getResource("openFile.gif"));
    image2 = new ImageIcon(simulateurpimsm.Frame1.class .getResource("closeFile.gif"));
    image3 = new ImageIcon(simulateurpimsm.Frame1.class .getResource("help.gif"));

    //setIconImage(Toolkit.getDefaultToolkit().createImage(Frame1.class.getResource("[Your Icon]")));
    contentPane = (JPanel) this .getContentPane();
    titledBorder1 = new TitledBorder("");
    titledBorder2 = new TitledBorder("");
    contentPane.setLayout(borderLayout1);
    this .setSize(new Dimension(693, 577));
    this .setTitle(this .frameTitle);
    statusBar.setText(" ");
    jMenuFile.setText("Fichier");
    jMenuFileExit.setText("Quitter");
    jMenuFileExit.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Aide");
    jMenuHelpAbout.setText("A propos");
    jMenuHelpAbout.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jMenuItemNew.setText("Nouveau");
    jMenuItemNew.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemNew_actionPerformed(e);
      }
    });
    jMenuItemOpen.setText("Ouvrir");
    jMenuItemOpen.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemOpen_actionPerformed(e);
      }
    });
    jMenuItemSave.setText("Enregistrer");
    jMenuItemSave.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemSave_actionPerformed(e);
      }
    });
    jMenuItemSaveAs.setActionCommand("EnregistrerSous");
    jMenuItemSaveAs.setText("Enregistrer Sous");
    jMenuItemSaveAs.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemSaveAs_actionPerformed(e);
      }
    });
    jMenuSimul.setText("Simulateur");
    jMenuItemHelp.setEnabled(false );
    jMenuItemHelp.setText("Aide");
    jMenuItemGstGroupesActifs.setText("Liste des groupes Actifs ...");
    jMenuItemGstGroupesActifs.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemGstGroupesActifs_actionPerformed(e);
      }
    });
    jButton5.setIcon(image2);
    jButton5.setToolTipText("Close File");
    jButton6.setToolTipText("Open File");
    jButton6.setIcon(image1);
    jButton10.setToolTipText("Help");
    jButton10.setIcon(image3);
    jLabelStatus.setBorder(BorderFactory.createEtchedBorder());
    jLabelStatus.setToolTipText("");
    jLabelStatus.setText("Simulateur PIM SM");
    jPanel1.setLayout(borderLayout3);
    jLabelEvenement.setBackground(Color.lightGray);
    jLabelEvenement.setBorder(BorderFactory.createLineBorder(Color.black));
    jLabelEvenement.setHorizontalAlignment(SwingConstants.CENTER);
    jLabelEvenement.setText("Evenements:");
    jPanel2.setLayout(borderLayout2);
    ListeEvenements.setBackground(SystemColor.inactiveCaptionText);
    jScrollPane2.getViewport().setBackground(Color.white);
    DomaineGraphique.setBackground(new Color(234, 250, 229));
    DomaineGraphique.setAlignmentX((float ) 0.0);
    DomaineGraphique.setAlignmentY((float ) 0.0);
    DomaineGraphique.setBorder(BorderFactory.createLineBorder(Color.black));
    DomaineGraphique.addMouseListener(new java.awt.event.MouseAdapter()
    {
      public void mouseClicked(MouseEvent e)
      {
        DomaineGraphique_mouseClicked(e);
      }
    });
    DomaineGraphique.setLayout(xYLayout1);
    jMenuItemAddRouteur.setActionCommand("AjouterRouteur");
    jMenuItemAddRouteur.setText("Ajouter un routeur");
    jMenuItemAddRouteur.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemAddRouteur_actionPerformed(e);
      }
    });
    jMenuItemAddLan.setActionCommand("AjouterLan");
    jMenuItemAddLan.setText("Ajouter un lan");
    jMenuItemAddLan.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jMenuItemAddLan_actionPerformed(e);
```

```
    }
  });
  jMenuItemAddStation.setActionCommand("AjouterStation");
  jMenuItemAddStation.setText("Ajouter une station");
  jMenuItemAddStation.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItemAddStation_actionPerformed(e);
    }
  });
  jMenuEdit.setText("Edition");
  jMenuItemSupElement.setActionCommand("SuppressionElément");
  jMenuItemSupElement.setText("Suppression de l\'élément sélectionné");
  jMenuItemSupElement.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItemSupElement_actionPerformed(e);
    }
  });
  jMenuItemConfigElement.setActionCommand("ConfigurationElément");
  jMenuItemConfigElement.setText("Configuration de l\'élément sélectionné");
  jMenuItemConfigElement.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItemConfigElement_actionPerformed(e);
    }
  });
  jMenu1.setText("Gestion des évènements traités");
  jCheckBoxMenuItem1.setText("Ajout / Suppression d\'un Lan");
  jCheckBoxMenuItem2.setText("Ajout / suppression d\'un routeur");
  jCheckBoxMenuItem3.setText("Ajout / Suppression d\'une Station");
  jMenuItem7.setText("Ajout ...");
  jCheckBoxMenuItem4.setText("Configuration CBSR d\'un routeur");
  jCheckBoxMenuItem5.setText("Configuration CRP d\'un Routeur");
  jCheckBoxMenuItem6.setText("Election d\'un nouveau BSR");
  jCheckBoxMenuItem7.setText("Election d\'un nouveau RP");
  jCheckBoxMenuItem8.setText("Election d\'un nouveau DR");
  jCheckBoxMenuItem9.setText("Election des \"Assert Winner\" d\'un Lan");
  jMenuItem1.setToolTipText("");
  jMenuItem1.setText("Liste des éléments du domaine");
  jMenuItem1.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItem1_actionPerformed(e);
    }
  });
  jMenuItemAddPaP.setText("Ajouter une ligne PàP");
  jMenuItemAddPaP.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItemAddPaP_actionPerformed(e);
    }
  });
  jLabelDescription.setBackground(Color.lightGray);
  jLabelDescription.setBorder(BorderFactory.createLineBorder(Color.black));
  jLabelDescription.setHorizontalAlignment(SwingConstants.CENTER);
  jLabelDescription.setText("Description");
  description.setText("Pas d\'objet sélectionné");
  description.setToolTipText("");
  description.setBackground(UIManager.getColor("inactiveCaptionText"));
  jSplitPane2.setOrientation(JSplitPane.VERTICAL_SPLIT);
  jPanel4.setLayout(borderLayout4);
  jPanel3.setLayout(borderLayout5);
  jMenuItemActivationGroupe.setText("Activation d\'un groupe");
  jMenuItemActivationGroupe.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItemActivationGroupe_actionPerformed(e);
    }
  });
  jMenuItem2.setText("Initialisation de la liste des évenements");
  jMenuItem2.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jMenuItem2_actionPerformed(e);
    }
  });
  jMenuFile.add(jMenuItemNew);
  jMenuFile.add(jMenuItemOpen);
  jMenuFile.addSeparator();
  jMenuFile.add(jMenuItemSave);
  jMenuFile.add(jMenuItemSaveAs);
  jMenuFile.addSeparator();
  jMenuFile.add(jMenuFileExit);
  jMenuHelp.add(jMenuItemHelp);
  jMenuHelp.add(jMenuHelpAbout);
  jMenuBar1.add(jMenuFile);
  jMenuBar1.add(jMenuEdit);
  jMenuBar1.add(jMenuSimul);
  jMenuBar1.add(jMenuHelp);
  this .setJMenuBar(jMenuBar1);
  contentPane.add(statusBar, BorderLayout.WEST);
  contentPane.add(jToolBar, BorderLayout.NORTH);
  jToolBar.add(jButton6, null );
  jToolBar.add(jButton5, null );
  jToolBar.add(jButton10, null );
  contentPane.add(jLabelStatus, BorderLayout.SOUTH);
  contentPane.add(jSplitPane1, BorderLayout.CENTER);
  jSplitPane1.add(jPanel1, JSplitPane.LEFT);
  jPanel1.add(jScrollPane2, BorderLayout.CENTER);
  jSplitPane1.add(jPanel2, JSplitPane.RIGHT);
  jPanel2.add(jSplitPane2, BorderLayout.CENTER);
  jSplitPane2.add(jPanel3, JSplitPane.TOP);
  jPanel3.add(jLabelDescription, BorderLayout.NORTH);
  jPanel3.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(description, null );
  jSplitPane2.add(jPanel4, JSplitPane.BOTTOM);
  jPanel4.add(jLabelEvenement, BorderLayout.NORTH);
  jPanel4.add(jScrollPane3, BorderLayout.CENTER);
  jScrollPane3.getViewport().add(ListeEvenements, null );
  jScrollPane2.getViewport().add(DomaineGraphique, null );
  DomaineGraphique.add(ListeEvenements, null );
  jMenuSimul.add(jMenuItemAddStation);
  jMenuSimul.add(jMenuItemAddRouteur);
  jMenuSimul.add(jMenuItemAddLan);
  jMenuSimul.add(jMenuItemAddPaP);
  jMenuSimul.addSeparator();
  jMenuSimul.add(jMenuItemActivationGroupe);
  jMenuSimul.add(jMenuItemGstGroupesActifs);
  jMenuSimul.add(jMenuItem1);
  jMenuEdit.add(jMenuItemSupElement);
  jMenuEdit.add(jMenuItemConfigElement);
  jMenuEdit.addSeparator();
  jMenuEdit.add(jMenu1);
  jMenuEdit.add(jMenuItem2);
  jMenu1.add(jCheckBoxMenuItem2);
  jMenu1.add(jCheckBoxMenuItem1);
  jMenu1.add(jCheckBoxMenuItem3);
  jMenu1.add(jCheckBoxMenuItem4);
  jMenu1.add(jCheckBoxMenuItem5);
  jMenu1.add(jCheckBoxMenuItem6);
  jMenu1.add(jCheckBoxMenuItem7);
```

```
    jMenu1.add(jCheckBoxMenuItem8);
    jMenu1.add(jCheckBoxMenuItem9);
    jMenu1.add(jMenuItem7);
    jSplitPane1.setDividerLocation(500);
    jSplitPane2.setDividerLocation(200);

    // Initialisation de la simulation
    initSimulation();

  }

  /**
   * Menu Fichier | Quitter
   */
  public   void  jMenuFileExit_actionPerformed(ActionEvent e)
  {
    if  (this .okPourAbandon())
    {
      System.exit(0);
    }
  }

  /**
   * Menu Aide | A propos
   */
  public   void  jMenuHelpAbout_actionPerformed(ActionEvent e)
  {
    Frame1_AboutBox dlg = new Frame1_AboutBox(this );
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loc.y);
    dlg.setModal(true );
    dlg.show();
  }

  /**
   * Gestion du titre de la frame
   */
  private      void   setTitreFrame()
  {
    if  (this .nomFichier == null  )
    {
      this .setTitle(this .frameTitle);
    }
    else
    {
      this .setTitle(this .frameTitle + this .nomFichier);
    }
  }

  /**
   * Traitement d'un évenement propre à la fenêtre
   */
  protected   void  processWindowEvent(WindowEvent e)
  {
    super .processWindowEvent(e);

    // Si l'évenement  = Fermeture de la fenêtre
    if  (e.getID() == WindowEvent.WINDOW_CLOSING)
    {
      jMenuFileExit_actionPerformed(null );
    }
  }

  void  jCheckBoxMenuItemsimulationActive_actionPerformed(ActionEvent e)
  {
    if  (simulationActive)
      {
        simulationActive = false  ;
      }
    else
      {
       simulationActive = true ;
      }

    ListeEvenements.setText(ListeEvenements.getText() + "\n Activation Sélection: " + new Boolean(simulationActive) );
  }

  /**
   * Menu Simulateur / Gestion Groupe Actif
   * Information sur les groupes actifs du domaine
   */
  void  jMenuItemGstGroupesActifs_actionPerformed(ActionEvent e)
  {
    DialogInformation dlg = new DialogInformation();

    dlg.setTitle(new String(" Information sur les groupes multicasts actifs sur le domaine :") );
    dlg.jText.setText(this .domaine.toStringLstGroupeMulticast());

    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    dlg.setSize((int ) frmSize.getWidth()/2, (int ) frmSize.getHeight()/2 );
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loc.y);

    dlg.show();
  }

  /**
   * Menu Simulateur / Gestion éléments du domaine
   * Information sur les éléments du domaine
   */
  void  jMenuItem1_actionPerformed(ActionEvent e)
  {
    DialogInformation dlg = new DialogInformation();

    dlg.setTitle(new String(" Information sur les éléments du domaine :") );

    dlg.jText.setText(domaine.toStringLstElements());

    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    dlg.setSize((int ) frmSize.getWidth()/2, (int ) frmSize.getHeight()/2 );
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loc.y);

    dlg.show();
  }

  /**
   * Initialisation des Objets Graphiques
   */
  private      void  initObjGraphique()
  {
     this .gestionEvenement = new GestionEvenements();
     this .objGraphiqueConfig = new ObjGraphique();
     this .objInterfaceGraphiqueConfig = new ObjInterfaceGraphique();

     this .gestionEvenement.initClass(this .ListeEvenements);

     this .objGraphiqueConfig.initClass(this .DomaineGraphique, this .description, this .objSelectionne, this .jLabelStatus,
this .domaine, this ,
                                        this .imgStation, this .imgRouteur, this .imgLan, this .imgPaP);
     this .objInterfaceGraphiqueConfig.initClass(this .DomaineGraphique, this .description, this .jLabelStatus, this .imgInterface
);

  }
```

```
  /**
   * Initialisation de la simulation
   */
  private       void  initSimulation()
  {
    domaine = new Domaine();

    this .domaineGph = new Vector();

    // Initialisation des objets graphiques
    this .initObjGraphique();

    // initialisation des groupes multicast
    initGroupeMulticast();

    jScrollPane3.getViewport().add(ListeEvenements, null );
  }

  /**
   * Initialisation des groupe Multicast
   */
  private       void  initGroupeMulticast()
  {
    // Pas réalisé pour lors de la période de Débugage
  }

  /**
   * Ouverture du fichier référencé par this.nomFichier
   */
  private       void  ouvrirFichier()
  {
    // Affichage de l'action dans la StatusBar
    String strStatus = this .jLabelStatus.getText();
    this .jLabelStatus.setText("Ouverture du domaine  : " + this .nomFichier);

    if  (this .nomFichier != null )
    {
      try
      {
        // Ouverture du fichier
        FileInputStream file = new FileInputStream(this .nomFichier);

        // Initialisation de la lecture d'objet
        ObjectInputStream objStream = new ObjectInputStream(file);

        // Récupération de l'entête du fichier
        String entFichier = null ;
        try
        {
          entFichier = (String) objStream.readObject();
        }
        catch  (Exception e)
        {
          // Le fichier n'est pas compatible.
          JOptionPane.showMessageDialog(this , "Le fichier n'est pas compatible !") ;

          // Fermeture du fichier
          objStream.close();

          this .jLabelStatus.setText(strStatus);
          return ;
        }

        // Si l'entête est correcte
        if  (entFichier != null )
        {
          if  (entFichier.equals(this .enteteFichier) )
          {
            // Chargement du domaine
            try
            {
              this .domaine = (Domaine) objStream.readObject();
            }
            catch  (Exception e)
            {
              // Le fichier n'est pas compatible.
              JOptionPane.showMessageDialog(this , "Impossible de charger le domaine !") ;

              // Fermeture du fichier
              objStream.close();

              this .jLabelStatus.setText(strStatus);
              return ;
            }

            // Initialisation des objets graphiques
            this .DomaineGraphique.removeAll();
            this .DomaineGraphique.repaint();

            this .description.setText(new String (""));
            this .ListeEvenements.setText(new String (""));

            this .domaineGph = new Vector();

            this .initObjGraphique();

            // Chargement du domaine graphique
            try
            {
               this .domaineGph = (Vector) objStream.readObject();
            }
            catch  (Exception e)
            {
              // Le fichier n'est pas compatible.
              JOptionPane.showMessageDialog(this , "Impossible de charger la représentation graphique du domaine !\n" + e) ;

              // Fermeture du fichier
              objStream.close();

              this .jLabelStatus.setText(strStatus);
              return ;
            }

            // Affichage du titre
            this .setTitreFrame();

            // Le domaine chargé en mémoire est identique au fichier.
            this .dirty = false ;

            JOptionPane.showMessageDialog(this , "Le domaine est chargé !") ;
          }
        }
      }
      catch  (Exception e)
      {
        // Erreur d'ouverture du fichier
        JOptionPane.showMessageDialog(this , "Problème lors de l'ouverture du fichier !") ;
      }
    }

    this .jLabelStatus.setText(strStatus);
  }

  /**
   * Sauvegarde du domaine dans le fichier
   */
  private    boolean    sauverFichier()
  {
    // Affichage de l'action dans la StatusBar
```

```
    String strStatus = this .jLabelStatus.getText();
    String nomAffiche = new String("");
    if  (this .nomFichier != null )
    {
      nomAffiche = " dans " + this .nomFichier;
    }
    this .jLabelStatus.setText("Sauvegarde du domaine dans " + nomAffiche );

    // Si le nom du fichier n'est pas défini, le définir
    if  (this .nomFichier == null )
    {
      boolean  res = sauverFichierComme();

      this .jLabelStatus.setText(strStatus);
      return  res;
    }

    try
    {
      // Ouvrir le fichier
      FileOutputStream file = new FileOutputStream (this .nomFichier);

      // Initialisation de la lecture d'Objet
      ObjectOutputStream objStream = new ObjectOutputStream(file);

      // Sauvegarde de l'entête
      try
      {
        objStream.writeObject(this .enteteFichier);
      }
      catch  (Exception e)
      {
        JOptionPane.showMessageDialog(this , "Problème lors de la sauvegarde de l'entête !") ;

        // Fermeture du fichier
        objStream.flush();
        objStream.close();

        this .jLabelStatus.setText(strStatus);
        return  false ;
      }

      // Sauvegarde du domaine
      try
      {
         objStream.writeObject(this .domaine);
      }
      catch  (Exception e)
      {
        JOptionPane.showMessageDialog(this , "Problème lors de la sauvegarde du domaine !") ;

        // Fermeture du fichier
        objStream.flush();
        objStream.close();

        this .jLabelStatus.setText(strStatus);
        return  false ;
      }

      // Sauvegarde du domaine graphique
      try
      {
         objStream.writeObject(this .domaineGph);
      }
      catch  (Exception e)
      {
        JOptionPane.showMessageDialog(this , "Problème lors de la sauvegarde de la représentation graphique du domaine !") ;

        // Fermeture du fichier
        objStream.flush();
        objStream.close();

        this .jLabelStatus.setText(strStatus);
        return  false ;
      }

      // Fermeture du fichier
      objStream.flush();
      objStream.close();

      // Le domaine chargé est identique au fichier
      this .dirty = false ;

      this .jLabelStatus.setText(strStatus);

      JOptionPane.showMessageDialog(this , "Le domaine est sauvegardé !") ;

      return  true ;
    }
    catch  (IOException e)
    {
      JOptionPane.showMessageDialog(this , "Problème de fichier !") ;

      this .jLabelStatus.setText(strStatus);
      return  false ;
    }
  }

  /**
   * Sauvegarde du domaine avec choix du nom du fichier
   */
  private  boolean  sauverFichierComme()
  {
    // Donner le choix du fichier à l'utilisateur
    if  (JFileChooser.APPROVE_OPTION == jFileChooser1.showSaveDialog(this ))
    {
      // Définir le nom du fichier
      this .nomFichier = jFileChooser1.getSelectedFile().getPath();

      return  sauverFichier();
    }
    else
    {
      this .repaint();
      return  false ;
    }
  }

  /**
   * L'utilisateur est-il d'accord d'abandonner
   * les éventuelles modifications effectuées sur le domaine ?
   *
   * ( Si besoin, sauvegarde des modifications avant de continuer )
   */
  boolean  okPourAbandon()
  {
    if  (!dirty)
    {
      return  true ;
    }

    int  value =  JOptionPane.showConfirmDialog(this , "Sauver les modifications ?",
                                          "Simulateur PIM SM", JOptionPane.YES_NO_CANCEL_OPTION) ;

    switch  (value)
    {
       case  JOptionPane.YES_OPTION:       // Sauver les modifications
         return  sauverFichier();
       case  JOptionPane.NO_OPTION:        // Abandonner les modifications
```

```
        return true ;
      case JOptionPane.CANCEL_OPTION:    // Annuler la demande
      default :
        return false ;
    }
  }

  /**
   * Menu Simulateur / Ajouter un routeur
   */
  void jMenuItemAddRouteur_actionPerformed(ActionEvent e)
  {
      // Affichage de l'action dans la StatusBar
      String strStatus = this .jLabelStatus.getText();
      this .jLabelStatus.setText("Ajout d'un routeur ");

      // Création du routeur dans le domaine
      Routeur routeur = new Routeur();
      domaine.addElement(routeur);

      // Création de sa représentation graphique
      ObjGraphique og = new ObjGraphique();
      og.initInstance(og.ROUTEUR, ""+routeur, routeur);
      og.addObject();

      // Ajout de cet objet à la liste des objets de cette frame
      this .domaineGph.addElement(og);

      // Le domaine est modifié
      this .dirty = true ;

      this .jLabelStatus.setText(strStatus);
  }

  /**
   * Menu Simulateur / Ajouter un Lan
   */
  void jMenuItemAddLan_actionPerformed(ActionEvent e)
  {
      // Affichage de l'action dans la StatusBar
      String strStatus = this .jLabelStatus.getText();
      this .jLabelStatus.setText("Ajout d'un lan ");

      // Création du lan dans le domaine
      Lan lan = new Lan();
      domaine.addElement(lan);

      // Création de sa représentation graphique
      ObjGraphique og = new ObjGraphique();
      og.initInstance(og.LAN, ""+lan, lan);
      og.addObject();

      // Ajout de cet objet à la liste des objets de cette frame
      this .domaineGph.addElement(og);

      // Le domaine est modifié
      this .dirty = true ;

      this .jLabelStatus.setText(strStatus);
  }

  /**
   * Menu Simulateur / Ajouter une Station
   */
  void jMenuItemAddStation_actionPerformed(ActionEvent e)
  {
      // Affichage de l'action dans la StatusBar
      String strStatus = this .jLabelStatus.getText();
      this .jLabelStatus.setText("Ajout d'une station ");

      // Création de la station dans le domaine
      Station station = new Station();
      domaine.addElement(station);

      // Création de sa représentation graphique
      ObjGraphique og = new ObjGraphique();
      og.initInstance(og.STATION, ""+station, station);
      og.addObject();

      // Ajout de cet objet à la liste des objets de cette frame
      this .domaineGph.addElement(og);

      // Le domaine est modifié
      this .dirty = true ;

      this .jLabelStatus.setText(strStatus);
  }

  /**
   * Menu Simulateur / Ajouter une ligne PàP
   */
  void jMenuItemAddPaP_actionPerformed(ActionEvent e)
  {
      // Affichage de l'action dans la StatusBar
      String strStatus = this .jLabelStatus.getText();
      this .jLabelStatus.setText("Ajout d'une ligne PàP ");

      // Création de la liaison PaP dans le domaine
      PaP pap = new PaP();
      domaine.addElement(pap);

      // Création de sa représentation graphique
      ObjGraphique og = new ObjGraphique();
      og.initInstance(og.PAP, "", pap);
      og.addObject();

      // Ajout de cet objet à la liste des objets de cette frame
      this .domaineGph.addElement(og);

      // Le domaine est modifié
      this .dirty = true ;

      this .jLabelStatus.setText(strStatus);
  }

  /**
   * Menu Fichier / Nouveau
   */
  void jMenuItemNew_actionPerformed(ActionEvent e)
  {
    if (this .okPourAbandon())
    {
      // initialisation de la simulation
      this .initSimulation();

      // initialisation du nom de fichier et du titre de la fenêtre
      this .nomFichier = null ;
      this .setTitreFrame();

      // Initialisation des objets graphiques
      this .DomaineGraphique.removeAll();
      this .DomaineGraphique.repaint();

      this .description.setText(new String (""));
      this .ListeEvenements.setText(new String (""));

      this .domaineGph = new Vector();

      this .initObjGraphique();
```

```
    }
  }

  /**
   * Menu Fichier / Ouvrir
   */
  void jMenuItemOpen_actionPerformed(ActionEvent e)
  {
    if (this .okPourAbandon())
    {
      if (JFileChooser.APPROVE_OPTION == jFileChooser1.showOpenDialog(this ))
      {
        //Affichage du nom du fichier dans le titre de la fenêtre
        this .nomFichier = new String("" + jFileChooser1.getSelectedFile().getPath());

        // ouverture et chargement du fichier
        this .ouvrirFichier();
      }
    }
  }

  /**
   * Menu Fichier / Enregistrer
   */
  void jMenuItemSave_actionPerformed(ActionEvent e)
  {
    this .sauverFichier();
  }

  /**
   * Menu Fichier / Enregistrer Sous
   */
  void jMenuItemSaveAs_actionPerformed(ActionEvent e)
  {
    this .sauverFichierComme();
  }

  /**
   * Gestion d'un click sur le domaine graphique
   */
  void DomaineGraphique_mouseClicked(MouseEvent e)
  {
    this .description.setText(domaine.toStringLstElements());

    ObjGraphique og = this .getObjSelectionne();
    if (og != null )
    {
      og.desactive();
    }
  }

  /**
   * Obtention de l'élément sélectionné
   */
  public      ObjGraphique getObjSelectionne()
  {
    if (objGraphiqueConfig != null )
    {
      return  objGraphiqueConfig.objSelectionne;
    }

    return  null ;
  }

  /**
   * Menu Edition / Suppression de l'élément
   */
  void jMenuItemSupElement_actionPerformed(ActionEvent e)
  {
    ObjGraphique objselect = this .objGraphiqueConfig.objSelectionne;
    // Si un objet est actuellement sélectionné
    if ( objselect != null )
    {
      // Suppression de l'objet du domaine
      if (objselect.representation != null )
      {
        if ( objselect.representation instanceof    ElementDomaine)
        {
          ElementDomaine ed = (ElementDomaine) objselect.representation;
          this .domaine.removeElement(ed);
        }
      }

      // suppression des interface graphique de l'objet graphique sélectionné
      this .objInterfaceGraphiqueConfig.removeAllFromObjGraphique(objselect);

      // Supression de l'objet graphique sélectionné
      this .objGraphiqueConfig.objSelectionne = null ;
      this .domaineGph.removeElement(objselect);

      // Suppression de sa représentation graphique
      this .DomaineGraphique.remove(objselect.objGraphique);
      this .DomaineGraphique.updateUI();
    }
  }

  /**
   * Menu Edition / Configuration de l'élément
   */
  void jMenuItemConfigElement_actionPerformed(ActionEvent e)
  {
    this .objGraphiqueConfig.configObjGraphiqueSelected();
  }

  /**
   * Menu Simulateur / Activation d'un groupe multicast
   */
  void jMenuItemActivationGroupe_actionPerformed(ActionEvent e)
  {
      if (this .domaine != null )
      {
        // configuration de l'adresse IP du nouveau groupe
        AdresseIp ai = new AdresseIp();
        ai.setNextAdresseMulticast();

        // Activation de ce groupe
        this .domaine.activeGroupeMulticast(ai);

        // Affichage de la boite de dialogue de confirmation d'ajout
        JOptionPane.showMessageDialog(this , "Le nouveau groupe multicast est activé à l'adresse " + ai ) ;
      }
  }

  /**
   * Menu Edition / Initialisation de la liste des évenements
   */
  void jMenuItem2_actionPerformed(ActionEvent e)
  {
    if (this .gestionEvenement != null )
    {
      this .gestionEvenement.initLstEvenement();
    }
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author Dupuis Eric
 * @version 1.0
 */

package simulateurpimsm;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import com.borland.jbcl.layout.*;

/**
 * Boite de Dialogue 'ABOUT'
 */
public class Frame1_AboutBox extends JDialog implements ActionListener
{

  JPanel panel1 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JButton button1 = new JButton();
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  String product = "Simulateur PIM SM";
  String version = "1.0";
  String copyright = "Copyright (c) 2000";
  String comments = "Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)";
  XYLayout xYLayout1 = new XYLayout();
  public Frame1_AboutBox(Frame parent)
  {
    super (parent);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    pack();
  }
  /**Component initialization*/
  private void jbInit() throws Exception
  {
    //imageLabel.setIcon(new ImageIcon(Frame1_AboutBox.class.getResource("[Your Image]")));
    this .setTitle("About");
    setResizable(false );
    panel1.setLayout(xYLayout1);
    label1.setFont(new java.awt.Font("Dialog", 3, 36));
    label1.setForeground(Color.blue);
    label1.setText(product);
    label2.setText("Version 1.0 Béta");
    label3.setText("Copyright (c) 2001 (Dupuis Eric)");
    label4.setText(comments);
    button1.setText("Ok");
    button1.addActionListener(this );
    this .getContentPane().add(panel1, BorderLayout.NORTH);
    panel1.add(insetsPanel1, new XYConstraints(0, 263, 536, -1));
    insetsPanel1.add(button1, null );
    panel1.add(label2, new XYConstraints(102, 189, 168, -1));
    panel1.add(label3, new XYConstraints(102, 206, 298, -1));
    panel1.add(label4, new XYConstraints(102, 223, -1, -1));
    panel1.add(label1, new XYConstraints(100, 69, 360, 52));
  }
  /**Overridden so we can exit when window is closed*/
  protected void processWindowEvent(WindowEvent e)
  {
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
    {
      cancel();
    }
    super .processWindowEvent(e);
  }
  /**Close the dialog*/
  void cancel()
  {
    dispose();
  }

  /**Close the dialog on a button event*/
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == button1)
    {
      cancel();
    }
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm;

import java.awt.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;

import simulateurpimsm.domaine.*;
import java.util.Vector;
import java.awt.event.*;
import javax.swing.event.*;

/**
 * Fenêtre de dialogue spécialisée pour la gestion de la configuration
 * d'un routeur actif sur le domaine
 */
public class ConfigRouteur extends JDialog
{
  JPanel panel1 = new JPanel();
  JLabel nomRouteur = new JLabel();
  XYLayout xYLayout1 = new XYLayout();
  JPanel jPanel1 = new JPanel();
  JCheckBox isCbsr = new JCheckBox();
  JCheckBox isCrp = new JCheckBox();
  JPanel jPanel2 = new JPanel();
  JLabel jLabel2 = new JLabel();
  XYLayout xYLayout2 = new XYLayout();
  JScrollPane jScrollPanel = new JScrollPane();
  JList listeConnexion = new JList();
  JCheckBox isConnected = new JCheckBox();
  JLabel jLabelVia = new JLabel();
  JLabel interf = new JLabel();

  // Objet du simulateur
  ObjGraphique objGraphique = null ;
  Routeur routeur = null ;

  Vector connexion = null ;
  int connexionSelect;


  /**
   * Constructeur
   */
  public ConfigRouteur(Frame frame, String title, boolean modal)
  {
    super (frame, title, modal);
    try
    {
      jbInit();
      pack();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }

  public ConfigRouteur()
  {
    this (null , "", false );
  }

  /**
   * Initialisation de la frame
   */
  void jbInit() throws Exception
  {
    panel1.setLayout(xYLayout1);
    nomRouteur.setFont(new java.awt.Font("Dialog", 1, 12));
    nomRouteur.setBorder(BorderFactory.createEtchedBorder());
    nomRouteur.setHorizontalAlignment(SwingConstants.CENTER);
    nomRouteur.setHorizontalTextPosition(SwingConstants.CENTER);
    nomRouteur.setText("Rt xx");
    jPanel1.setBorder(BorderFactory.createLoweredBevelBorder());
    isCbsr.setText("Candidat BSR");
    isCbsr.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        isCbsr_actionPerformed(e);
      }
    });
    isCrp.setText("Cadidat RP");
    isCrp.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        isCrp_actionPerformed(e);
      }
    });
    jPanel2.setBorder(BorderFactory.createLoweredBevelBorder());
    jPanel2.setLayout(xYLayout2);
    jLabel2.setText("Liste des Connexions :");
    isConnected.setText("Connexion établie");
    isConnected.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        isConnected_actionPerformed(e);
      }
    });
    jLabelVia.setText("via:");
    interf.setText("I xx (Adr: xx.xx.xx.xx)");
    listeConnexion.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    listeConnexion.addListSelectionListener(new javax.swing.event.ListSelectionListener()
    {
      public void valueChanged(ListSelectionEvent e)
      {
        listeConnexion_valueChanged(e);
      }
    });
    getContentPane().add(panel1);
    panel1.add(nomRouteur, new XYConstraints(4, 6, 120, 41));
    panel1.add(jPanel1, new XYConstraints(158, 6, 227, 41));
    jPanel1.add(isCbsr, null );
    jPanel1.add(isCrp, null );
    panel1.add(jPanel2, new XYConstraints(6, 61, 381, 223));
    jPanel2.add(jLabel2, new XYConstraints(4, 6, 232, 26));
    jPanel2.add(jScrollPanel, new XYConstraints(7, 39, 220, 174));
    jPanel2.add(isConnected, new XYConstraints(233, 41, -1, -1));
    jPanel2.add(jLabelVia, new XYConstraints(233, 66, -1, -1));
    jPanel2.add(interf, new XYConstraints(254, 64, 122, 20));
    jScrollPanel.getViewport().add(listeConnexion, null );
  }

  /**
   * Initialisation du contenu
   * (Doit-être réalisé avant l'affichache de la frame)
   */
  void initRouteur (ObjGraphique obj)
```

```
{
  if (obj != null )
  {
    this .objGraphique = obj;
    this .routeur = (Routeur) obj.representation;
  }

  if (routeur != null )
  {
    // Nom du routeur
    this .nomRouteur.setText(""+ routeur);


    // Configuration de Cbsr et CRP
    this .isCbsr.setSelected(routeur.isCBSR());
    this .isCrp.setSelected(routeur.isCRP());

    // Liste de toutes les connexions du domaine
    this .connexion = new Vector();
    int i = 0;
    int j = routeur.getDomaine().getNbrConnexion();
    while ( i < j )
    {
      // Selection de l'élément actif à traiter
      Connexion c = routeur.getDomaine().getConnexion(i);

      // Ajout s'il est une station
      if (c != null )
      {
          this .connexion.addElement(c);
      }

      // connexion suivante
      i = i + 1;
    }

    // Configuration des Objets listes
    this .listeConnexion.setListData(this .connexion);
    this .isConnected.disable();

    // Sélection de la première connexion de la liste
    this .connexionSelect = 0;
    this .listeConnexion.setSelectedIndex(0);
  }
}

/**
 * Gestion de la demande de changement d'état de CBSR
 */
void isCbsr_actionPerformed(ActionEvent e)
{
  // S'il s'agit d'une activation
  if (this .isCbsr.isSelected())
  {
      routeur.getDomaine().setCBSR(routeur, true );
      this .objGraphique.description.setText(routeur.toStringConfig());
  }
  // s'il s'agit d'une désactivation
  else
  {
      routeur.getDomaine().setCBSR(routeur, false );
      this .objGraphique.description.setText(routeur.toStringConfig());
  }
}

/**
 * Gestion de la demande de changement d'état de CRP
 */
void isCrp_actionPerformed(ActionEvent e)
{
  // S'il s'agit d'une Activation
  if (this .isCrp.isSelected())
  {
      routeur.getDomaine().setCRP(routeur, true );
      this .objGraphique.description.setText(routeur.toStringConfig());
  }
  // s'il s'agit d'une désactivation
  else
  {
      routeur.getDomaine().setCRP(routeur, false );
      this .objGraphique.description.setText(routeur.toStringConfig());
  }
}

/**
 * Gestion de la demande de changement d'état de la connexion
 */
void isConnected_actionPerformed(ActionEvent e)
{
  // Obtention de la connexion référencée
  int index = this .listeConnexion.getSelectedIndex();
  if (index < this .connexion.size() )
  {
    Connexion c = (Connexion) this .connexion.elementAt(index);
    if ( c != null )
    {
      // S'il s'agit d'une Activation
      if (this .isConnected.isSelected())
      {
        // Connexion des éléments du domaine
        routeur.getDomaine().connectElements(routeur, c);

        // Création de l'interface graphique entre les deux éléments graphiques
        ObjGraphique connexionGraphique = ObjGraphique.getObjGraphique(c);
        if (connexionGraphique != null )
        {
          ObjInterfaceGraphique.add(this .objGraphique, connexionGraphique);
        }
      }
      // s'il s'agit d'une désactivation
      else
      {
          // obtention de l'interface
          Interface interf = routeur.getInterfaceTo((Connexion) this .connexion.elementAt(index));

          // Suppression de l'interface graphique
          ObjInterfaceGraphique oig = ObjInterfaceGraphique.getInterfaceGraphiqueDe(interf, this .objGraphique);
          if (oig != null )
          {
            ObjInterfaceGraphique.remove(oig);
          }

          // Déconnexion des éléments du domaine
          routeur.getDomaine().deconnectElements(routeur, (Connexion) this .connexion.elementAt(index));
      }
    }

    this .objGraphique.description.setText(routeur.toStringConfig());
  }
}

/**
 * Gestion du changement de la connexion sélectionnée
 */
void listeConnexion_valueChanged(ListSelectionEvent e)
{
  if (this .listeConnexion.isSelectionEmpty())
  {
      this .isConnected.disable();
```

```
        this .jLabelVia.setVisible(false );
        this .interf.setVisible(false );
    }
    else
    {
        this .isConnected.enable();

        Connexion c = null ;

        // obtention de la connexion référencée
        int  index = this .listeConnexion.getSelectedIndex();
        if (index < this .connexion.size())
        {
          c = (Connexion) this .connexion.elementAt(this .listeConnexion.getSelectedIndex());
        }

        // Configuration de isConnected
        Interface result = null ;
        if ( c != null )
        {
          result = this .routeur.getInterfaceTo(c);
        }

        if (result == null )
        {
          this .isConnected.setSelected(false );
          this .jLabelVia.setVisible(false );
          this .interf.setVisible(false );
          // ...
        }
        else
        {
          this .isConnected.setSelected(true );
          this .jLabelVia.setVisible(true );
          this .interf.setVisible(true );
          this .interf.setText("" + result);
          // ...
        }
    }
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm;

import java.awt.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;
import java.awt.event.*;
import javax.swing.event.*;

import simulateurpimsm.domaine.*;
import java.util.Vector;

/**
 * Fenêtre de dialogue spécialisée pour la gestion de la configuration
 * d'une station active sur le domaine
 */
public class ConfigStation extends JDialog
{
  JPanel panel1 = new JPanel();
  XYLayout xYLayout1 = new XYLayout();
  JLabel nomStation = new JLabel();
  JPanel jPanel1 = new JPanel();
  JButton SuppressionInterface = new JButton();
  XYLayout xYLayout2 = new XYLayout();
  JLabel nomInterface = new JLabel();
  JLabel jLabel3 = new JLabel();
  JLabel nomConnexion = new JLabel();
  JPanel jPanel2 = new JPanel();
  JPanel jPanel3 = new JPanel();
  JLabel jLabel4 = new JLabel();
  XYLayout xYLayout3 = new XYLayout();
  JLabel jLabel5 = new JLabel();
  XYLayout xYLayout4 = new XYLayout();
  JList listFluxEmis = new JList();
  JCheckBox isEmis = new JCheckBox();
  JList listeFluxRecus = new JList();
  JCheckBox isRecu = new JCheckBox();
  JScrollPane jScrollPanel = new JScrollPane();
  JScrollPane jScrollPane2 = new JScrollPane();
  JLabel adresseIp = new JLabel();
  JLabel jLabel6 = new JLabel();
  TitledBorder titledBorder1;

  // Objet du simulateur
  ObjGraphique objGraphique = null ;
  Station station = null ;
  Interface interf = null ;

  Vector fluxEmis = null ;
  Vector fluxRecus = null ;
  int emisSelect;
  int recuSelect;

  /**
   * Constructeur
   */
  public ConfigStation(Frame frame, String title, boolean modal)
  {
    super(frame, title, modal);
    try
    {
      jbInit();
      pack();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }


  public ConfigStation()
  {
    this (null , "", false );
  }

  /**
   * Initialisation de la frame
   */
  void jbInit() throws Exception
  {
    titledBorder1 = new TitledBorder("");
    panel1.setLayout(xYLayout1);
    nomStation.setFont(new java.awt.Font("Dialog", 1, 12));
    nomStation.setBorder(titledBorder1);
    nomStation.setHorizontalAlignment(SwingConstants.CENTER);
    nomStation.setHorizontalTextPosition(SwingConstants.CENTER);
    nomStation.setText("St. xx");
    jPanel1.setBorder(BorderFactory.createLoweredBevelBorder());
    jPanel1.setLayout(xYLayout2);
    SuppressionInterface.setHorizontalTextPosition(SwingConstants.CENTER);
    SuppressionInterface.setMargin(new Insets(2, 4, 2, 4));
    SuppressionInterface.setText("suppression");
    SuppressionInterface.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        SuppressionInterface_actionPerformed(e);
      }
    });
    nomInterface.setText("I.xx");
    jLabel3.setText("Connectée à :");
    nomConnexion.setText("Lan xx");
    jPanel2.setBorder(BorderFactory.createLoweredBevelBorder());
    jPanel2.setLayout(xYLayout3);
    jPanel3.setBorder(BorderFactory.createLoweredBevelBorder());
    jPanel3.setLayout(xYLayout4);
    jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel4.setHorizontalTextPosition(SwingConstants.CENTER);
    jLabel4.setText("Gestion des flux émis :");
    jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel5.setHorizontalTextPosition(SwingConstants.CENTER);
    jLabel5.setText("Gestion des flux réceptionnés :");
    isEmis.setText("Emission du flux");
    isEmis.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        isEmis_actionPerformed(e);
      }
    });
    isRecu.setText("Réception du flux");
    isRecu.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        isRecu_actionPerformed(e);
      }
    });
    listeFluxRecus.setBorder(BorderFactory.createEtchedBorder());
```

```
    listeFluxRecus.setSelectionBackground(Color.lightGray);
    listeFluxRecus.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    listeFluxRecus.addListSelectionListener(new javax.swing.event.ListSelectionListener()
    {
      public void valueChanged(ListSelectionEvent e)
      {
        listeFluxRecus_valueChanged(e);
      }
    });
    listFluxEmis.setBorder(BorderFactory.createEtchedBorder());
    listFluxEmis.setSelectionBackground(Color.lightGray);
    listFluxEmis.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    listFluxEmis.addListSelectionListener(new javax.swing.event.ListSelectionListener()
    {
      public void valueChanged(ListSelectionEvent e)
      {
        listFluxEmis_valueChanged(e);
      }
    });
    adresseIp.setText("xx.xx.xx.xx");
    jLabel6.setText("Adr :");
    getContentPane().add(panel1);
    panel1.add(jPanel3, new XYConstraints(4, 178, 393, 111));
    jPanel3.add(jLabel5, new XYConstraints(0, 0, 347, -1));
    jPanel3.add(jScrollPane2, new XYConstraints(2, 17, 263, 89));
    jPanel3.add(isRecu, new XYConstraints(267, 22, -1, -1));
    jScrollPane2.getViewport().add(listeFluxRecus, null );
    panel1.add(jPanel1, new XYConstraints(106, 9, 290, 50));
    jPanel1.add(nomConnexion, new XYConstraints(88, 21, 77, -1));
    jPanel1.add(jLabel3, new XYConstraints(1, 21, -1, -1));
    jPanel1.add(nomInterface, new XYConstraints(2, 3, 50, -1));
    jPanel1.add(jLabel6, new XYConstraints(53, 3, -1, -1));
    jPanel1.add(adresseIp, new XYConstraints(88, 3, 80, -1));
    jPanel1.add(SuppressionInterface, new XYConstraints(180, 6, 98, 29));
    panel1.add(jPanel2, new XYConstraints(4, 64, 393, 106));
    jPanel2.add(jLabel4, new XYConstraints(1, 0, 346, -1));
    jPanel2.add(isEmis, new XYConstraints(266, 21, -1, -1));
    jPanel2.add(jScrollPane1, new XYConstraints(3, 16, 262, 84));
    panel1.add(nomStation, new XYConstraints(3, 10, 100, 28));
    jScrollPane1.getViewport().add(listFluxEmis, null );
  }

  /**
   * Initialisation du contenu
   * (Doit-être réalisé avant l'affichache de la frame)
   */
  void   initStation (ObjGraphique obj)
  {
    if  (obj != null )
    {
      this .objGraphique = obj;
      station = (Station) obj.representation;
    }

    if  (station != null )
    {
      // Nom de la station
      this .nomStation.setText(""+ station);

      // Interface (Nom, Connexion et adresse)
      interf = station.getInterface(0);
      if  (interf != null )
      {
        this .nomInterface.setText("" + interf);
        this .nomConnexion.setText("" + interf.getConnexion());
        this .adresseIp.setText("" + interf.getAdresse());
      }
      else
      {
        this .nomInterface.setText(new String("Non Défini"));
        this .nomConnexion.setText(new String("Non Défini"));
        this .adresseIp.setText(new String("Non Défini"));
      }

      // Liste de toutes les station du domaine (pour la creation des flux (S,G)
      Vector lstStation = new Vector();
      int  i = 0;
      int  j = station.getDomaine().getNbrElementActif();
      while  ( i < j )
      {
        // Selection de l'élément actif à traiter
        ElementActif ea = station.getDomaine().getElementActif(i);

        // Ajout s'il est une station
        if  (ea != null )
        {
          if  (ea instanceof  Station)
          {
            lstStation.addElement(ea);
          }
        }

        i = i + 1;
      }


      // Liste des flux émissibles et réceptibles
      fluxEmis = new Vector();
      fluxRecus = new Vector();
      i = 0;
      j = station.getDomaine().getNbrGroupeActif();
      while  ( i < j )
      {
        //Obtention du groupe actif à traiter
        GroupeMulticast gm = station.getDomaine().getGroupeActif(i);

        if  ( gm != null )
        {
          // Création d'un nouveau flux
          Fluxmulticast flux = new Fluxmulticast();
          flux.setGroupe(gm);

          // Ajout de ce flux aux deux listes
          fluxEmis.addElement(flux);
          fluxRecus.addElement(flux);

          // Gestion des flux spécifiques réceptibles.
          int  i2 = 0;
          int  j2 = lstStation.size();
          while  ( i2 < j2 )
          {
            // Obtention de la station
            Station st = (Station) lstStation.elementAt(i2);
            if  (st != null )
            {
              // création d'un flux spécifique
              Fluxmulticast fluxSp = new Fluxmulticast();
              fluxSp.setSource(st);
              fluxSp.setGroupe(gm);

              //Ajout à la liste des flux réceptibles
              fluxRecus.addElement(fluxSp);
            }

            // Station suivante
            i2 = i2 + 1;
          }
        }
```

```
      // traiter le groupe actif suivant
      i = i + 1;
    }

    // Configuration des Objets listes
    this .listFluxEmis.setListData(fluxEmis);
    this .listeFluxRecus.setListData(fluxRecus);

    this .isRecu.disable();
    this .isEmis.disable();

    // Sélection des premiers flux dans les listes
    this .emisSelect = 0;
    this .listFluxEmis.setSelectedIndex(0);

    this .recuSelect = 0;
    this .listeFluxRecus.setSelectedIndex(0);
  }
}

/**
 * Gestion de la demande de suppression de l'interface
 */
void  SuppressionInterface_actionPerformed(ActionEvent e)
{
  if  ( this .interf != null );
  {
    // Déconnexion des éléments et suppression de l'interface
    if  ( (this .interf.getElementActif() != null ) && (this .interf.getConnexion() != null ) )
    {
      // déconnexion des éléments du domaine
      station.getDomaine().deconnectElements(this .interf.getElementActif(), this .interf.getConnexion());

      // suppression de l'interface graphique
      ObjInterfaceGraphique oig = ObjInterfaceGraphique.getInterfaceGraphiqueDe(this .interf, this .objGraphique);
      if  (oig != null )
      {
        ObjInterfaceGraphique.remove(oig);
      }

      // mise à jour de cette boite de dialogue
      this .nomInterface.setText(new String("Non Défini"));
      this .nomConnexion.setText(new String("Non Défini"));
      this .adresseIp.setText(new String("Non Défini"));

      this .objGraphique.description.setText(station.toStringConfig());
    }
  }
}

/**
 * Gestion de la demande de changement d'état d'un flux Emis
 */
void  isEmis_actionPerformed(ActionEvent e)
{
  Fluxmulticast fm = null ;

  // obtention du flux référencé
  int  index = this .listFluxEmis.getSelectedIndex();
  if  (index < this .fluxEmis.size())
  {
    fm = (Fluxmulticast) this .fluxEmis.elementAt(index);
  }

  if  ( fm != null )
  {
    // S'il s'agit d'une sélection du flux
    if  (this .isEmis.isSelected())
    {
      // Emettre le flux sélectionné
      station.addSource(fm.getGroupe());
    }
    // s'il s'agit d'une désélection
    else
    {
      // Fin d'émission du flux sélectionné
      station.removeSource(fm.getGroupe());
    }

    this .objGraphique.description.setText(station.toStringConfig());
  }
}

/**
 * Gestion du changement du flux reçu sélectionné
 */
void  listFluxEmis_valueChanged(ListSelectionEvent e)
{
  if  (this .listFluxEmis.isSelectionEmpty())
  {
    this .isEmis.disable();
  }
  else
  {
    this .isEmis.enable();

    Fluxmulticast result = null ;

    // obtention du flux référencé
    int  index = this .listFluxEmis.getSelectedIndex();
    if  ( index < this .fluxEmis.size())
    {
      Fluxmulticast fm = (Fluxmulticast) this .fluxEmis.elementAt(index);

      // Configuration de la valeur de isEmis
      result = this .station.getFlux(Station.EMIS, fm.getGroupe(), fm.getSource());
    }

    if  (result == null )
    {
      this .isEmis.setSelected(false );
    }
    else
    {
      this .isEmis.setSelected(true );
    }
  }
}

/**
 * Gestion de la demande de changement d'état d'un flux Recus
 */
void  isRecu_actionPerformed(ActionEvent e)
{
  Fluxmulticast fm = null ;

  // obtention du flux référencé
  int  index = this .listeFluxRecus.getSelectedIndex();
  if  (index < this .fluxRecus.size() )
  {
    fm = (Fluxmulticast) this .fluxRecus.elementAt(index);
  }

  if  ( fm != null )
  {
    // S'il s'agit d'une sélection
    if  (this .isRecu.isSelected())
    {
      // recevoir le flux sélectionné
```

```
        if  (fm.getSource() == null )
        {
          station.addRecever(fm.getGroupe());
        }
        else
        {
          station.addRecever(fm.getSource(), fm.getGroupe());
        }
      }
      // s'il s'agit d'une désélection
      else
      {
        // terminer la réception du flux sélectionné
        if  (fm.getSource() == null )
        {
          station.removeRecever(fm.getGroupe());
        }
        else
        {
          station.removeRecever(fm.getSource(), fm.getGroupe());
        }
      }

      this .objGraphique.description.setText(station.toStringConfig());
    }
  }

  /**
   * Gestion du changement du flux émis sélectionné
   */
  void  listeFluxRecus_valueChanged(ListSelectionEvent e)
  {
    if  (this .listeFluxRecus.isSelectionEmpty())
    {
        this .isRecu.disable();
    }
    else
    {
        this .isRecu.enable();

        Fluxmulticast result = null ;

        // obtention du flux référencé
        int  index = this .listeFluxRecus.getSelectedIndex();
        if  ( index < this .fluxRecus.size() )
        {
          Fluxmulticast fm = (Fluxmulticast) this .fluxRecus.elementAt(index);

          // Configuration de la valeur de isEmis
          result = this .station.getFlux(Station.RECUS, fm.getGroupe(), fm.getSource());
        }

        if  (result == null )
        {
          this .isRecu.setSelected(false );
        }
        else
        {
          this .isRecu.setSelected(true );
        }
    }
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */
package simulateurpimsm;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;

/**
 * Boîte de dialogue d'information
 * (Est utilisée pour afficher de l'information sous format texte)
 */
public class DialogInformation extends JDialog implements ActionListener
{
  JPanel panel1 = new JPanel();
  JScrollPane jScrollPanel = new JScrollPane();
  JTextArea jText = new JTextArea();
  GridLayout gridLayout1 = new GridLayout();

  /**
   * Constructeur
   */
  public DialogInformation(Frame frame, String title, boolean modal)
  {
    super(frame, title, modal);

    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
      pack();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }

  public DialogInformation()
  {
    this (null , "", false );
  }

  /**
   * Initialisation de la frame
   */
  void jbInit() throws Exception
  {
    panel1.setLayout(gridLayout1);
    getContentPane().add(panel1);
    panel1.add(jScrollPanel, null );
    jScrollPanel.getViewport().add(jText, null );
  }

  /**
   * fermeture de la boîte de dialogue
   */
  public void actionPerformed(ActionEvent e)
  {
      dispose();
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm;

import java.awt.*;
import javax.swing.*;
import simulateurpimsm.domaine.message.*;
import simulateurpimsm.domaine.*;

/**
 * Classe spécialisée pour la gestion des événements dans le domaine
 *( Elle sert d'interface entre les éléments du domaine et l'interface utilisateur.)
 *
 * Une instance de celle-ci doit-être en premier lieu configuré par l'interface utilisateur (Frame1)
 * Ensuite, chaque objet du domaine a le loisir de créer une nouvelle instance de cette classe et de l'utiliser pour loguer un
évenement.
 *
 * Remarque :
 * Dans cette version du simulateur, cette classe est loin d'être optimalisée
 * et nécessitera une gestion des évènements plus complexes dans les prochaines versions.
 * ( E A: - implémentation d'un mode pas à pas
 *        - configuration des événements que l'utilisateur désire loguer
 *        - Gestion de plus grand panel d'événements
 * )
 */
public class GestionEvenements
{
  // Zone texte d'impression des évenements
  static JTextArea lstEvenement = null ;

  // Numérotation des événements traités
  static int numEvenement = 0;

  /**
   * Constructeur
   */
  public GestionEvenements()
  {
  }

  /**
   * Initialisation de la classe
   * (Doit-être réalisé avant l'utilisation par les éléments du domaine)
   */
  public void initClass(JTextArea texteArea)
  {
    this.lstEvenement = texteArea;

    initLstEvenement();
  }

  /**
   * initalisation de la gestion des événements
   */
  public void initLstEvenement()
  {
    this.lstEvenement.setText("");
    this.numEvenement = 0;
  }

  /**
   * Affichage d'un événement
   */
  private void afficheEvenement(String evenement)
  {
    if (this.lstEvenement != null )
    {
      this.numEvenement += 1;

      String lstEv = this.lstEvenement.getText();

      lstEv = lstEv + "\n" + new Integer(this.numEvenement) + " => " + evenement;

      this.lstEvenement.setText(lstEv);
    }
  }


  /**
   * Evénement message en entrée ou en sortie sur un routeur
   *
   * direction = true    pour une arrivée de message
   * direction = false   pour une expédition de message
   */
  public void setEvenementMessage(Routeur routeur, boolean direction, Interface interf, Message msg)
  {
    String sDirection = new String(routeur + " : Expédition d'un ");
    if ( direction )
    {
      sDirection = new String(routeur + " : Réception d'un ");
    }

    String type = null ;
    String sMsg = new String("");

    if (msg instanceof Register )
    {
      type = new String("Register");

      Register m = (Register) msg;

      sMsg = "(" + m.getSourceFlux() + ", " + m.getGroupe() + ")"
             + " à destination de " + m.getDestination();

      this.afficheEvenement(sDirection + type + sMsg + " Via " + interf);
      return ;
    }

    if (msg instanceof RegisterStop)
    {
      type = new String("RegisterStop");

      RegisterStop m = (RegisterStop) msg;
      sMsg = "(" + m.getSourceFlux() + ", " + m.getGroupe() + ")"
             + " à destination de " + m.getDestination();

      this.afficheEvenement(sDirection + type + sMsg + " Via " + interf);
      return ;
    }

    if (msg instanceof JoinG)
    {
      type = new String("Join");

      JoinG m = (JoinG) msg;
      sMsg = "(*, " + m.getGroupe() + ")";


      this.afficheEvenement(sDirection + type + sMsg + " Via " + interf);
      return ;
    }

    if (msg instanceof JoinSG)
```

```
    {
      type = new String("Join");

      JoinSG m = (JoinSG) msg;
      sMsg = "(" + m.getSourceFlux() + ", " + m.getGroupe() + ")";


      this .afficheEvenement(sDirection + type  + sMsg + " Via " + interf);
      return ;
    }

    if  (msg instanceof    PruneG)
    {
      type = new String("Prune");

      PruneG m = (PruneG) msg;
      sMsg = "(*, " + m.getGroupe() + ")";

      this .afficheEvenement(sDirection + type  + sMsg + " Via " + interf);
      return ;
    }

    if  (msg instanceof    PruneSG)
    {
      type = new String("Prune");

      PruneSG m = (PruneSG) msg;
      sMsg = "(" + m.getSourceFlux() + ", " + m.getGroupe() + ")";

      this .afficheEvenement(sDirection + type  + sMsg + " Via " + interf);
      return ;
    }

    if  (msg instanceof    PruneSGRpt)
    {
      type = new String("PruneSGRpt");

      PruneSGRpt m = (PruneSGRpt) msg;
      sMsg = "(" + m.getSourceFlux() + ", " + m.getGroupe() + ")";

      this .afficheEvenement(sDirection + type  + sMsg + " Via " + interf);
      return ;
    }

    if  (msg instanceof    MsgMulticast)
    {
      type = new String("Message Multicast");

      MsgMulticast m = (MsgMulticast) msg;
      sMsg = " émis par " + m.getSource() + " vers " + m.getGroupe();

      this .afficheEvenement(sDirection + type  + sMsg + " Via " + interf);
      return ;
    }

    if  (msg instanceof    Message)
    {
      type = new String("message");

      sMsg = new String("");

      this .afficheEvenement(sDirection + type  + sMsg + " Via " + interf);
      return ;
    }
}


 /**
  * Evénement message en entrée ou en sortie sur un routeur
  *
  * type = true    pour un IGMP Qwery
  * type = false   pour un IGMP Prune
  */
  public  void  setEvenementIgmp(Lan lan, Routeur dr, boolean  type, Station source, AdresseIp groupe)
 {
    String sType = new String("IGMP Prune ");
    if  ( type )
    {
       sType =  new String("IGMP Qwery");
    }

    String src = new String("*");
    if  (source != null )
    {
      src = "" + source;
    }

    this .afficheEvenement(sType + " sur " + dr + " en provenance de " + lan + " pour le flux(" + src + ", " + groupe +")" );
    return ;


 }
}
```

```
package simulateurpimsm;

import java.awt.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.awt.event.*;
import com.borland.dbswing.*;
import com.borland.dx.dataset.*;

/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

public class GroupesActifsDlgBox extends JDialog
{
  JPanel panel1 = new JPanel();
  XYLayout xYLayout1 = new XYLayout();
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JPanel jPanel1 = new JPanel();
  XYLayout xYLayout2 = new XYLayout();
  JComboBox jComboBox1 = new JComboBox();
  JLabel jLabel1 = new JLabel();
  JColorChooser jColorChooser = new JColorChooser();
  JLabel jLabel2 = new JLabel();
  JTextField jTextFieldAdrGroupe = new JTextField();
  JLabel jLabel3 = new JLabel();
  JButton jButton3 = new JButton();

  public GroupesActifsDlgBox(Frame frame, String title, boolean modal)
  {
    super(frame, title, modal);
    try
    {
      jbInit();
      pack();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }

  public GroupesActifsDlgBox()
  {
    this(null, "", false);
  }
  void jbInit() throws Exception
  {
    panel1.setLayout(xYLayout1);
    jButton1.setText("Ok");
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jButton1_actionPerformed(e);
      }
    });
    jButton2.setText("Annuler");
    jButton2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        jButton2_actionPerformed(e);
      }
    });
    jPanel1.setLayout(xYLayout2);
    jPanel1.setBorder(BorderFactory.createLoweredBevelBorder());
    jLabel1.setText("Groupe Actif :");
    jColorChooser.setBorder(BorderFactory.createLoweredBevelBorder());
    jColorChooser.setToolTipText("");
    jLabel2.setText("Adresse :");
    jLabel3.setToolTipText("");
    jLabel3.setText("Couleur :");
    jButton3.setText("Ajouter Groupe");
    this.getContentPane().add(panel1, BorderLayout.CENTER);
    panel1.add(jComboBox1, new XYConstraints(112, 25, 187, -1));
    panel1.add(jLabel1, new XYConstraints(27, 27, -1, -1));
    panel1.add(jPanel1, new XYConstraints(7, 55, 516, 406));
    jPanel1.add(jLabel2, new XYConstraints(11, 16, -1, -1));
    jPanel1.add(jTextFieldAdrGroupe, new XYConstraints(70, 15, 128, -1));
    jPanel1.add(jLabel3, new XYConstraints(11, 48, -1, -1));
    jPanel1.add(jColorChooser, new XYConstraints(69, 45, -1, -1));
    panel1.add(jButton3, new XYConstraints(317, 22, -1, -1));
    panel1.add(jButton1, new XYConstraints(533, 396, 77, -1));
    panel1.add(jButton2, new XYConstraints(533, 434, -1, -1));
  }

  void jButton1_actionPerformed(ActionEvent e)
  {
    dispose();
  }

  void jButton2_actionPerformed(ActionEvent e)
  {
    dispose();
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.XYConstraints;
import javax.swing.border.LineBorder;
import javax.swing.border.EtchedBorder;
import java.io.*;

import simulateurpimsm.domaine.*;
import java.util.Vector;

/**
 * Classe encapsulant un objet du domaine (Station, Routeur, PàP ou Lan)
 * et s'occupant de sa représentation graphique
 * ainsi que de l'interfaçage de celui-ci avec l'utilisateur.
 * (Affichage et actions sur évènements souris)
 */
public class ObjGraphique implements Serializable
{
  // Constantes
  public static final int STATION = 1;
  public static final int ROUTEUR = 2;
  public static final int LAN = 3;
  public static final int PAP = 4;

  // Position par défaut d'un objet
  static int defPosX = 10;
  static int defPosY = 10;

  // taille par défaut des objets
  static int defStationTailleX = 80;
  static int defStationTailleY = 50;

  static int defRouteurTailleX = 70;
  static int defRouteurTailleY = 30;

  static int defLanTailleX = 80;
  static int defLanTailleY = 60;

  static int defPapTailleX = 12;
  static int defPapTailleY = 12;

  // Référence des objets de la frame utilisés pour l'affichage graphique
  static JPanel domaineGraphique = null ;
  static JTextArea description = null ;
  static ObjGraphique objSelectionne = null ;
  static JLabel statusBar = null ;
  static Domaine domaine = null ;
  static Frame frame = null ;

  // Icone des objets
  static ImageIcon imageStation = null ;
  static ImageIcon imageRouteur = null ;
  static ImageIcon imageLan = null ;
  static ImageIcon imagePap = null ;


  // String utilisé pour le stockage temporaire du contenu de la StatusBar
  String statusValue = null ;

  // Objet représenté
  Object representation = null ;

  // Image de cette instance
  ImageIcon image = null ;

  // label de l'objet
  String label = null ;

  // Label Graphique représentant l'objet
  JLabel objGraphique = null ;

  // Position de l'obj dans l'interface
  int posX;
  int posY;

  // Taille de l'obj
  int tailleX;
  int tailleY;

  // Liste d'interfaces graphiques reliées à cette instance de l'objet Graphique
  Vector lstObjInterface = null ;

  // données utilisées lors d'un drag
  Point departPoint = null ;
  boolean isDrag = false ;

  /**
   * Constructeur
   */
  public ObjGraphique()
  {
    // Définition de la taille de l'obj
    posX = defPosX;
    posY = defPosY;
  }

  /**
   * Initialisation de la classe
   */
  public void initClass (JPanel newDomaineGraphique, JTextArea newDescription, ObjGraphique newObjSelectionne, JLabel
newStatusBar, Domaine newDomaine, Frame newFrame,
                         ImageIcon newImageStation, ImageIcon newImageRouteur, ImageIcon newImageLan, ImageIcon newImagePap
)
  {
    domaineGraphique = newDomaineGraphique;
    description      = newDescription;
    objSelectionne   = newObjSelectionne;
    statusBar        = newStatusBar;
    domaine          = newDomaine;
    frame            = newFrame;

    imageStation     = newImageStation;
    imageRouteur     = newImageRouteur;
    imageLan         = newImageLan;
    imagePap         = newImagePap;
  }

  /**
   * Initialisation de l'intance
   */
  public void initInstance(int type, String newLabel, Object newObject)
  {
    this.posX = this.defPosX;
    this.posY = this.defPosY;

    this.label = newLabel;
    this.representation = newObject;
```

```
    if  ( type == this .STATION)
    {
      this .image = this .imageStation;
      this .tailleX = this .defStationTailleX;
      this .tailleY = this .defStationTailleY;

      return ;
    }

    if  ( type == this .ROUTEUR)
    {
      this .image = this .imageRouteur;
      this .tailleX = this .defRouteurTailleX;
      this .tailleY = this .defRouteurTailleY;

      return ;
    }

    if  ( type == this .LAN)
    {
      this .image = this .imageLan;
      this .tailleX = this .defLanTailleX;
      this .tailleY = this .defLanTailleY;

      return ;
    }

    if  ( type == this .PAP)
    {
      this .image = this .imagePap;
      this .tailleX = this .defPapTailleX;
      this .tailleY = this .defPapTailleY;

      return ;
    }
  }

  /**
   * Configuration de l'objet sélectionné
   */
  void  configObjGraphiqueSelected()
  {
    ObjGraphique objselect = this .objSelectionne;
    if  ( objselect != null )
    {
      // Si l'objet sélectionné est une station
      if  ( objselect.representation instanceof    Station)
      {
            // Création de la boite de dialogue
            ConfigStation dlg = new ConfigStation(frame, new String("configuration Station"), false );

            // positionnement de la boite de dialogue
            Dimension dlgSize = dlg.getPreferredSize();
            Dimension frmSize = frame.getSize();
            Point loc = frame.getLocation();
            dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loc.y);

            // initialisation de la boite de dialogue
            dlg.initStation(objselect);

            // Affichage de la boite de dialogue
            dlg.show();
      }

      // Si l'objet sélectionné est un routeur
      if  ( objselect.representation instanceof    Routeur)
      {
            // Création de la boite de dialogue
            ConfigRouteur dlg = new ConfigRouteur(frame, new String("configuration Routeur"), false );

            // positionnement de la boite de dialogue
            Dimension dlgSize = dlg.getPreferredSize();
            Dimension frmSize = frame.getSize();
            Point loc = frame.getLocation();
            dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loc.y);

            // initialisation de la boite de dialogue
            dlg.initRouteur(objselect);

            // Affichage de la boite de dialogue
            dlg.show();
      }
    }
  }


  /**
   * Gestion du label
   */
  public      void   setLabel(String newLabel)
  {
    label = newLabel;
  }

  public      String  getLabel()
  {
    return  label;
  }

  /**
   * Gestion de l'objet représenté
   */
  public      void   setObject(Object newObject)
  {
    representation = newObject;
  }

  public      Object  getObject()
  {
    return  representation;
  }

  /**
   * Gestion de la taille et de la position
   */
  public      void   setPosition(int  x, int  y)
  {
    this .posX = x;
    this .posY = y;
  }

  public      void   setTaille(int  x, int  y)
  {
    this .tailleX = x;
    this .tailleY = y;
  }

  public      int  getPositionX()
  {
    return  posX;
  }

  public      int  getPositionY()
  {
    return  posY;
  }

  public      int  getTailleX()
```

```
{
  return  tailleX;
}

public      int  getTailleY()
{
  return  tailleY;
}

/**
 * Recherche de l'objet graphique représentant un élément du domaine
 */
static   ObjGraphique getObjGraphique(ElementDomaine elementDomaine)
{
    int  i = 0;
    int  j = 0;
    ObjGraphique result = null ;

    if  (elementDomaine != null )
    {
     if  ( ((Frame1) frame).domaineGph != null )
     {
        j =  ((Frame1) frame).domaineGph.size();
     }
    }

    while  (i < j)
    {
      ObjGraphique og = (ObjGraphique) ((Frame1) frame).domaineGph.elementAt(i);
      if  (og != null )
      {
        if  (og.representation == elementDomaine)
        {
          return  og;
        }
      }
      i = i + 1;
    }

  return  null ;
}

/**
 * Activation de l'objet
 */
void  active()
{
    // si un objet est déjà sélectionné
    if  (this .objSelectionne != null )
    {
      this .objSelectionne.desactive();
    }

    this .objSelectionne = this ;
    this .objGraphique.setBorder(new LineBorder(Color.darkGray));
}

/**
 * Désactivation de l'objet
 */
void  desactive()
{
  this .objGraphique.setBorder(new LineBorder(Color.lightGray));
}


/**
 * Ajout de la représentation graphique de l'objet dans le Panel
 */
public   void  addObject()
{
  objGraphique = new JLabel();
  objGraphique.setHorizontalAlignment(SwingConstants.CENTER);
  objGraphique.setHorizontalTextPosition(SwingConstants.RIGHT);
  objGraphique.setIcon(image);
  objGraphique.setText(label);
  objGraphique.setBorder(new LineBorder(Color.lightGray));
  objGraphique.setCursor(new Cursor (Cursor.CROSSHAIR_CURSOR));

  // Gestion des Events
  objGraphique.addMouseMotionListener(new java.awt.event.MouseMotionAdapter()
  {
    public   void  mouseMoved(MouseEvent e)
    {
      objGraphique_mouseMoved(e);
    }
    public   void  mouseDragged(MouseEvent e)
    {
      objGraphique_mouseDragged(e);
    }
  });

  objGraphique.addMouseListener(new java.awt.event.MouseAdapter()
  {
    public   void  mousePressed(MouseEvent e)
    {
      objGraphique_mousePressed(e);
    }
    public   void  mouseReleased(MouseEvent e)
    {
      objGraphique_mouseReleased(e);
    }
    public   void  mouseClicked(MouseEvent e)
    {
      objGraphique_mouseClicked(e);
    }
    public   void  mouseEntered(MouseEvent e)
    {
      objGraphique_mouseEntered(e);
    }
    public   void  mouseExited(MouseEvent e)
    {
      objGraphique_mouseExited(e);
    }

  });

  // Ajout de cet objet au domaine
  this .domaineGraphique.add(objGraphique, new XYConstraints(posX, posY, tailleX, tailleY));

  // Activation de cet objet
  this .active();
}

/**
 * Gestion de la liaison entre deux objets
 */
boolean    liaison(ObjGraphique from, ObjGraphique to )
{
  // Si le domaine n'est pas configuré
  if  ( this .domaine == null  )
  {
      // Affichage de la boite de dialogue de problème
      JOptionPane.showMessageDialog(this .domaineGraphique, "Le domaine doit être configuré !") ;
      return  false ;
  }

  // Si un des deux éléments graphiques n'est pas défini
  if  ( (from == null ) || (to == null ) )
```

```
    {
        // Affichage de la boite de dialogue de problème
        JOptionPane.showMessageDialog(this .domaineGraphique, "Les deux éléments graphique doivent être configuré !") ;
        return  false ;
    }

    // Si un des deux éléments du domaine n'est pas défini
    if  ( (from.representation == null ) || (to.representation == null ) )
    {
        // Affichage de la boite de dialogue de problème
        JOptionPane.showMessageDialog(this .domaineGraphique, "Les deux éléments du domaine doivent être configuré !" ) ;
        return  false ;
    }

    boolean  result = false ;

    // Ne connecter qu'un élément actif à une connexion
    if  ( from.representation instanceof  ElementActif )
    {
      if  (to.representation instanceof  Connexion)
      {
        ElementActif ea = (ElementActif) from.representation;
        Connexion c = (Connexion) to.representation;

        // liaison des éléments
        if ( this .domaine.connectElements(ea, c) )
        {
          // Si connexion réussie, Création et affichage de l'interface Graphique
          ObjInterfaceGraphique.add(from, to);
        }
        else
        {
            // Affichage de la boite de dialogue de problème
            JOptionPane.showMessageDialog(this .domaineGraphique, "Les deux éléments du domaine n'ont pas été reliés !" ) ;
            return  false ;
        }
      }
    }
    else
    {
      if  (to.representation instanceof  ElementActif)
      {
        ElementActif ea = (ElementActif) to.representation;
        Connexion c = (Connexion) from.representation;

        // liaison des éléments
        if  (this .domaine.connectElements(ea, c))
        {
          // Si connexion réussie, Création et affichage de l'interface Graphique
          ObjInterfaceGraphique.add(to, from);
        }
        else
        {
            // Affichage de la boite de dialogue de problème
            JOptionPane.showMessageDialog(this .domaineGraphique, "Les deux éléments du domaine n'ont pas été reliés !" ) ;
            return  false ;
        }
      }
    }

    return  true ;
  }

  /**
   * Gestion des Events souris : MousePressed
   */
  void  objGraphique_mousePressed(MouseEvent e)
  {
    // Si le bouton de droite est utilisé
    if  ( (e.getModifiers() & e.BUTTON3_MASK)== e.BUTTON3_MASK )
    {
      // Voir s'il s'agit d'une demande de liaison avec l'élément actuellement sélectionné
      if  ( this .objSelectionne != null  )
      {
        liaison (this , this .objSelectionne);
      }
    }

    // retenir la position de départ d'un drag éventuel
    this .departPoint = e.getPoint();

    // Initialiser la possibilité d'un drag
    this .isDrag = false ;
  }

  /**
   * Gestion des Events souris : MouseDragged
   */
  void  objGraphique_mouseDragged(MouseEvent e)
  {
    // retenir qu'un drag est en cours
    this .isDrag = true ;
  }

  /**
   * Gestion des Events souris : MouseReleased
   */
  void  objGraphique_mouseReleased(MouseEvent e)
  {
    if  ( this .isDrag )
    {
      Point arriveePoint = e.getPoint();
      int  x =  arriveePoint.x - this .departPoint.x;
      int  y =  arriveePoint.y - this .departPoint.y;

      // reposition de l'objet
      Point localisation = this .objGraphique.getLocation();
      posX = localisation.x + x;
      posY = localisation.y + y;

      this .domaineGraphique.remove(this .objGraphique);
      this .domaineGraphique.add(objGraphique, new XYConstraints(posX, posY, tailleX, tailleY));
      this .domaineGraphique.updateUI();

      // Repositionnement des interfaces graphiques de cet objet
      ObjInterfaceGraphique.afficheAllInterfaces(this );


      // Acticvation de l'objet
      this .active();
    }
  }

  void  objGraphique_mouseMoved(MouseEvent e)
  {
//   if (this.description != null)
//    {
//      this.description.setText("(mouseMoved) Description de " + this.representation);
//    }
  }

  void  objGraphique_mouseClicked(MouseEvent e)
  {
    if  (this .description != null )
    {
      if  (this .representation instanceof  ElementDomaine)
      {
        ElementDomaine ed = (ElementDomaine) this .representation;
        this .description.setText(ed.toStringConfig());
```

```
      }

      // Activation de l'objet
      this .active();

      // Si Double click => affichage config
      if (e.getClickCount() > 1 )
      {
        this .configObjGraphiqueSelected();
      }
    }
  }

  void objGraphique_mouseEntered(MouseEvent e)
  {
      // Sauvegarde de la valeur de la StatusBar
      this .statusValue = this .statusBar.getText();
      this .statusBar.setText("Objet domaine : " + this .representation);
  }

  void objGraphique_mouseExited(MouseEvent e)
  {
      // Reqstauration de la valeur de la StatusBar
      this .statusBar.setText(this .statusValue);
  }



/**                                                                    **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                    */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private   void writeObject(ObjectOutputStream s) throws IOException
  {
    // Sauvegarde des éléments du domaine représenté par cette instance
    s.writeObject(this .representation);

    // Sauvegarde du type d'icone utilisé pour cette instance
    if ( this .image == this .imageLan )
    {
              s.writeInt(this .LAN);
    }
    else
    { if ( this .image == this .imagePap )
      {
              s.writeInt(this .PAP);
      }
      else
      { if ( this .image == this .imageRouteur )
        {
              s.writeInt(this .ROUTEUR);
        }
        else
        {
              s.writeInt(this .STATION);
        }
      }
    }

    // Sauvegarde du label de cette instance
    s.writeObject(this .label);

    // Sauvegade de la taille et de la position de l'objet
    s.writeInt(this .posX);
    s.writeInt(this .posY);
    s.writeInt(this .tailleX);
    s.writeInt(this .tailleY);

    // Sauvegade de la liste d'interface graphique
    s.writeObject(this .lstObjInterface);
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private   void readObject(ObjectInputStream s) throws IOException, ClassNotFoundException
  {
    // Recupération des éléments du domaine représenté par cette instance
    this .representation = s.readObject();

    // Recupération du type d'icone utilisé pour cette instance
    int img = s.readInt();
    if ( img == this .LAN )
    {
              this .image = this .imageLan;
    }
    else
    { if ( img == this .PAP )
      {
              this .image = this .imagePap;
      }
      else
      { if ( img == this .ROUTEUR )
        {
              this .image = this .imageRouteur;
        }
        else
        {
              this .image = this .imageStation;
        }
      }
    }

    // Recupération du label de cette instance
    this .label = (String) s.readObject();

    // Recupération de la taille et de la position de l'objet
    this .posX     = s.readInt();
    this .posY     = s.readInt();
    this .tailleX  = s.readInt();
    this .tailleY  = s.readInt();

    // Récupération de la liste d'interface graphique
    this .lstObjInterface = (Vector) s.readObject();

    // Création de la représentation graphique
    this .addObject();
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import javax.swing.border.LineBorder;
import javax.swing.border.EtchedBorder;
import java.io.*;

import simulateurpimsm.domaine.*;
import java.util.Vector;

/**
 * Classe encaplusant une interface du domaine
 * et s'occupant de sa représentation graphique
 * ainsi que de l'interfaçage de celui-ci avec l'utilisateur.
 * (Affichage et actions sur évènements souris)
 */
public class ObjInterfaceGraphique implements Serializable
{
  // Interface représentée
  Interface interf = null ;

  // Représentation graphique de l'élément actif graphique
  ObjGraphique elementActifGraphique = null ;

  // Représentation graphique de la connexion graphique
  ObjGraphique connexionGraphique = null ;

  // Label Graphique représentant l'objet
  JLabel objGraphique = null ;
  JPanel ligneGraphique = null ;

  // Position de l'obj dans l'interface
  int posX;
  int posY;

  // Taille d'une interface
  static int tailleX = 10;
  static int tailleY = 10;

  // Référence des objets de la frame utilisés pour l'affichage graphique
  static JPanel domaineGraphique = null ;
  static JTextArea description = null ;
  static JLabel statusBar = null ;

  // String utilisé pour le stockage temporaire du contenu de la StatusBar
  String statusValue = null ;

  // repésentation graphique de l'interface
  static ImageIcon imageInterface = null ;


  /**
   * Constructeur
   */
  public ObjInterfaceGraphique()
  {
  }

  /**
   * Initialisation de la classe
   */
  public void initClass (JPanel newDomaineGraphique, JTextArea newDescription, JLabel newStatusBar, ImageIcon
newImageInterface)
  {
    domaineGraphique = newDomaineGraphique;
    description      = newDescription;
    statusBar        = newStatusBar;

    imageInterface    = newImageInterface;
  }

  /**
   * Ajout d'une nouvelle interface
   */
  static void add(ObjGraphique newElementActifGraphique, ObjGraphique newConnexionGraphique)
  {
    // si les éléments fournis ne sont valides
    if ( newElementActifGraphique != null ) {
    if ( newConnexionGraphique != null )
    {
      // Création de l'interface
      ObjInterfaceGraphique oig = new ObjInterfaceGraphique();

      // Configuration des objets Graphiques
      oig.elementActifGraphique = newElementActifGraphique;
      oig.connexionGraphique = newConnexionGraphique;
      oig.interf = ((ElementActif) oig.elementActifGraphique.representation).getInterfaceTo((Connexion) oig.connexionGraphique
.representation);

      // Ajout de cette instance aux deux objets graphiques
      oig.addToObjGraphique(newElementActifGraphique);
      oig.addToObjGraphique(newConnexionGraphique);

      // Création de la représentation graphique de cette instance
      oig.initInstanceGraphique();
    }}
  }

  /**
   * Initialisation de la représentation graphique de cet objet
   */
  void initInstanceGraphique()
  {
      this.objGraphique = new JLabel();
      this.objGraphique.setHorizontalAlignment(SwingConstants.CENTER);
      this.objGraphique.setHorizontalTextPosition(SwingConstants.RIGHT);
      this.objGraphique.setIcon(imageInterface);
      this.objGraphique.setBorder(new LineBorder(Color.lightGray));
      this.objGraphique.setCursor(new Cursor (Cursor.TEXT_CURSOR));

      // Configuration de la gestion de la souris pour cette instance
      this.objGraphique.addMouseListener(new java.awt.event.MouseAdapter()
      {
        public void mouseEntered(MouseEvent e)
        {
          objGraphique_mouseEntered(e);
        }
        public void mouseExited(MouseEvent e)
        {
          objGraphique_mouseExited(e);
        }
      });

      // Positionnement et affichage de cette instance
      this.afficheInterface();
  }
```

```
/**
 * Ajout de cette instance graphique d'une interface à un objet graphique
 */
void  addToObjGraphique(ObjGraphique toObj)
{
  if  ( toObj != null )
  {
    if  (toObj.lstObjInterface == null )
    {
      toObj.lstObjInterface = new Vector();
    }
    toObj.lstObjInterface.addElement(this );
  }
}

/**
 * Suppression de toutes les instances d'interfaces graphique d'un objet graphiques
 */
static   void  removeAllFromObjGraphique(ObjGraphique fromObj)
{
   int  i = 0;
   int  j = 0;

   if  (fromObj != null )
   {
    if  (fromObj.lstObjInterface != null )
    {
       j =  fromObj.lstObjInterface.size();
    }
   }

   while  (i < j)
   {
     ObjInterfaceGraphique oig = (ObjInterfaceGraphique) fromObj.lstObjInterface.elementAt(i);
     if  (oig != null )
     {
       // suppression de l'interface graphique de la liste des deux objets graphiques
       remove(oig);
     }

     // J est décrémenté et non i incrémenté car la méthode remove modifie la liste
     j =  fromObj.lstObjInterface.size();
   }
}

/**
 * Suppression d'une interface graphique
 */
static   void  remove(ObjInterfaceGraphique oldInterfaceGraphique)
{
 if  (oldInterfaceGraphique != null )
 {
    // suppression de la représentation graphique
    oldInterfaceGraphique.cacheInterface();

    // suppression de l'interface dans le premier objet Graphique
    oldInterfaceGraphique.removeFromObjGraphique(oldInterfaceGraphique.elementActifGraphique);

    // suppression de l'interface dans le deuxième objet Graphique
    oldInterfaceGraphique.removeFromObjGraphique(oldInterfaceGraphique.connexionGraphique);
 }
}

/**
 * Suppression de cette instance de l'interface dans un objet graphique
 */
void  removeFromObjGraphique(ObjGraphique fromObj)
{
   if  ( fromObj != null )
   {
     if  (fromObj.lstObjInterface != null )
     {
       fromObj.lstObjInterface.removeElement(this );
     }
   }
}

/**
 * Recherche d'une interface graphique représentant 'interface' et connectée à 'fromObj'
 */
static   ObjInterfaceGraphique getInterfaceGraphiqueDe(Interface interf, ObjGraphique fromObj)
{
   int  i = 0;
   int  j = 0;
   ObjInterfaceGraphique result = null ;

   if  (fromObj != null )
   {
    if  (fromObj.lstObjInterface != null )
    {
       j =  fromObj.lstObjInterface.size();
    }
   }

   while  (i < j)
   {
     ObjInterfaceGraphique oig = (ObjInterfaceGraphique) fromObj.lstObjInterface.elementAt(i);
     if  (oig != null )
     {
       if  (oig.interf == interf)
       {
         return  oig;
       }
     }

     i = i + 1;
   }

   return  null ;
}

/**
 * Affichage de cette instance de l'interface
 */
void  afficheInterface()
{
  // recalcul la position de l'objet
  this .positionne();

  // Affichage de la ligne de connexion
  this .afficheLigne();

  // Affichage de la représentation graphique de cette instance
  this .domaineGraphique.add(this .objGraphique, new XYConstraints(posX, posY, tailleX, tailleY));
  this .domaineGraphique.updateUI();
}

/**
 * Affichage de la ligne de connexion
 */
void  afficheLigne()
{
  Point start = new Point(this .posX+this .tailleX/2, this .posY+this .tailleY/2);
  Point end   = this .getPoint(this .connexionGraphique, this .elementActifGraphique);

  int  lngX = 0;
  int  lngY = 0;
```

```
    XYConstraints xyc = null ;
    boolean   direction = false  ;

    if  ( start.x < end.x )
    {
      lngX = end.x - start.x;
      if  ( start.y < end.y)
      {
        lngY = end.y - start.y;

        xyc = new XYConstraints (start.x, start.y,lngX , lngY);
        direction = true ;
      }
      else
      {
        lngY = start.y - end.y;

        xyc = new XYConstraints (start.x, start.y- lngY, lngX , lngY);
      }
    }
    else
    {
      lngX = start.x - end.x;
      if  ( start.y < end.y)
      {
        lngY = end.y - start.y;

        xyc = new XYConstraints (start.x-lngX, start.y, lngX , lngY);
      }
      else
      {
        lngY = start.y - end.y;

        xyc = new XYConstraints (start.x-lngX , start.y-lngY, lngX , lngY);
        direction = true ;
      }
    }

    // création du conteneur de la ligne de connexion
    this .ligneGraphique = new JPanel();
    this .ligneGraphique.setOpaque(false  );
    this .ligneGraphique.setLayout(new XYLayout() );
    this .domaineGraphique.add(this .ligneGraphique, xyc);

    // Affichage ligne de points
    int  i = 1;
    int  j = 30;
    while  ( i < j )
    {
      JLabel l = new JLabel(new String("."));

      l.setBorder(BorderFactory.createEtchedBorder(Color.lightGray, Color.magenta));

      float   x = 0;
      float   y = 0;

      if  (direction)
      {
        x = (lngX * i) / j;
        y = (lngY * i) / j;
      }
      else
      {
        x = (lngX * i) / j;
        y = (lngY * (j-i) ) / j;
      }

      this .ligneGraphique.add(l, new XYConstraints((int ) x, (int ) y, 3, 3));
      i = i + 1;
      }
    }

    /**
     * Suppression de l' affichage de cette instance de l'interface
     */
    void  cacheInterface()
    {
      // suppression des lignes de connexion
      if  (this .ligneGraphique != null )
      {
        this .domaineGraphique.remove(this .ligneGraphique);
      }

      // suppression de la représentation graphique de cette instance
      this .domaineGraphique.remove(this .objGraphique);
      this .domaineGraphique.updateUI();
    }

    /**
     * Affichage de toute les interfaces connectées à l'objet graphique fromObj
     */
    static   void  afficheAllInterfaces(ObjGraphique fromObj)
    {
      int  i = 0;
      int  j = 0;

      if  (fromObj != null  )
      {
        if  (fromObj.lstObjInterface != null )
        {
          j = fromObj.lstObjInterface.size();
        }
      }

      while  ( i < j)
      {
        //Obtention de l'interface
        ObjInterfaceGraphique oig = (ObjInterfaceGraphique) fromObj.lstObjInterface.elementAt(i);
        if  (oig != null )
        {
          oig.cacheInterface();
          oig.afficheInterface();
        }

        i = i + 1;
      }
    }

    /**
     * Calcul la position de l'interface sur la représentation graphique
     */
    void   positionne()
    {
      if  ( (this .elementActifGraphique != null ) && (this .connexionGraphique != null ) )
      {
        Point centre = ObjInterfaceGraphique.getPoint(this .elementActifGraphique, this .connexionGraphique);
        this .posX = centre.x - (this .tailleX / 2);
        this .posY = centre.y - (this .tailleY / 2);
      }
    }

    /**
     * Calcul des coordonées du point d'intersection entre
     * - le segment de droite entre les centres des objets 'start' et 'end'
     * - et le cadre de l'objet 'start'
     */
    static   private   Point getPoint (ObjGraphique start, ObjGraphique end)
    {
```

```
    int  pStartX = (start.posX + start.tailleX/ 2);
    int  pStartY = (start.posY + start.tailleY/ 2);

    int  pEndX   = (end.posX + end.tailleX/2);
    int  pEndY   = (end.posY + end.tailleY/2);

    int  distX   = pEndX - pStartX;
    if  ( distX == 0)
    {
      distX = 1;
    }

    int  distY   = pEndY - pStartY;
    if  ( distY == 0)
    {
      distY = 1;
    }

    Point point = null ;

    if  (distX > 0)
    {
      // A droite
      if  (distY > 0)
      {
        // En bas / droite
        if  ( (distX / start.tailleX) < (distY / start.tailleY) )
        {
          point = new Point ( ( pStartX + ( distX * start.tailleY/2) / distY)
                            , ( pStartY + start.tailleY/2)                    );
        }
        else
        {
           point = new Point ( ( pStartX + start.tailleX/2)
                             , ( pStartY + ( distY * start.tailleX/2) / distX) );
        }
      }
      else
      {
        distY = -distY;
        // En Haut / droite
        if  ( (distX / start.tailleX) < (distY / start.tailleY) )
        {
          point = new Point ( ( pStartX + ( distX * start.tailleY/2) / distY)
                            , ( pStartY - start.tailleY/2)                    );
        }
        else
        {
           point = new Point ( ( pStartX + start.tailleX/2)
                             , ( pStartY - ( distY * start.tailleX/2) / distX) );
        }
      }
    }
    else
    {
      distX = -distX;
      // A gauche
      if  (distY > 0)
      {
        // En bas / gauche
        if  ( (distX / start.tailleX) < (distY / start.tailleY) )
        {
          point = new Point ( ( pStartX - ( distX * start.tailleY/2) / distY)
                            , ( pStartY + start.tailleY/2)                    );
        }
        else
        {
           point = new Point ( ( pStartX - start.tailleX/2)
                             , ( pStartY + ( distY * start.tailleX/2) / distX) );
        }
      }
      else
      {
        distY = -distY;
        // En Haut / gauche
        if  ( (distX / start.tailleX) < (distY / start.tailleY) )
        {
          point = new Point ( ( pStartX - ( distX * start.tailleY/2) / distY)
                            , ( pStartY - start.tailleY/2)                    );
        }
        else
        {
           point = new Point ( ( pStartX - start.tailleX/2)
                             , ( pStartY - ( distY * start.tailleX/2) / distX) );
        }
      }
    }

    return  point;
  }

/**                                                                          **
 * --------  Méthodes utilisées pour la gestion de la souris--------------- *
 *                                                                          */

  void  objGraphique_mouseEntered(MouseEvent e)
  {
      // Sauvegarde de la valeur de la StatusBar
      this .statusValue = this .statusBar.getText();
      this .statusBar.setText("Interface : " + this .interf.toStringConfig());
  }

  void  objGraphique_mouseExited(MouseEvent e)
  {
      // Reqstauration de la valeur de la StatusBar
      this .statusBar.setText(this .statusValue);
  }


/**                                                                          **
 * --------  Méthodes utilisées pour la sérialisation --------------------- *
 *                                                                          */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private  void  writeObject(ObjectOutputStream s) throws  IOException
  {
    // Sauvegarde de l'interface représentés par cet instance+-
    s.writeObject(this .interf);

    // Sauvegarde des extrémités graphiques de cette interface
    s.writeObject(this .elementActifGraphique);
    s.writeObject(this .connexionGraphique);

    // sauvegarde de la position de cette interface graphique
    s.writeInt(this .posX);
    s.writeInt(this .posY);
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private  void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
  {
    // Recupération de l'interface représenté par cet instance
    this .interf = (Interface) s.readObject();

    // Recupération des extrémités graphiques de cette interface
```

```
    this .elementActifGraphique = (ObjGraphique) s.readObject();
    this .connexionGraphique = (ObjGraphique) s.readObject();

    // Recupération de la position de cette interface graphique
    this .posX = s.readInt();
    this .posY = s.readInt();

    // Création de la représentation graphique et affichage de celle-ci
    this .initInstanceGraphique();
//    this.afficheInterface();
  }
}
```

8.4.2. Les éléments du domaine

(Package simulateurpimsm.domaine)

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.io.*;

/**
 * Classe représentant une adrese IP
 * et gérant toutes les opérations pouvant être réalisées sur celle-ci
 * (Entre autre : - Attribution automatique des adresses de sous-réseau pour les connexions
 *                  - Attribution automatique des adresses pour les groupes multicast actifs
 *                - Les opérations de comparaison entre adresses
 */
public    class   AdresseIp implements   Serializable
{
   // Variables statiques permettant une attribution automatique de l'adresse d'un sous réseau (une connexion)
   private      static    int  nextAdrReseauByte1 = 10;
   private      static    int  nextAdrReseauByte2 = 10;
   private      static    int  nextAdrReseauByte3 = 0;

   // Variables statiques permettant une attribution automatique de l'adresse d'un nouveau groupe multicast
   private      static    int  nextAdrMulticastByte1 = 225;
   private      static    int  nextAdrMulticastByte2 = 10;
   private      static    int  nextAdrMulticastByte3 = 10;

   private      int  adrByte1 = 0;
   private      int  adrByte2 = 0;
   private      int  adrByte3 = 0;
   private      int  adrByte4 = 0;

   /**
   * Constructeur:
   */
   public   AdresseIp()
   {
   }

   public   AdresseIp(AdresseIp adrReseau)
   {
    setAdresse(adrReseau);
   }

  /**
   * Configure cette instance à l'adresse réseau suivante.
   *               Est utilisé par une instance de l'objet connexion pour définir son adresse réseau.
   */
   public        void  setNextAdresseReseau ()
   {
    nextAdrReseauByte3 = nextAdrReseauByte3 + 1;

    adrByte1 = nextAdrReseauByte1;
    adrByte2 = nextAdrReseauByte2;
    adrByte3 = nextAdrReseauByte3;
    adrByte4 = 1;
   }

  /**
   * Configure cette instance à l'adresse multicast IP suivante.
   *               Est utilisé par une instance de l'objet groupeMulticast pour définir son adresse de groupe.
   */
   public        void  setNextAdresseMulticast ()
   {
    nextAdrMulticastByte3 = nextAdrMulticastByte3 + 1;

    adrByte1 = nextAdrMulticastByte1;
    adrByte2 = nextAdrMulticastByte2;
    adrByte3 = nextAdrMulticastByte3;
    adrByte4 = 0;
   }

  /**
   * Configure cette instance à la même adresse que l'instance passée en paramètre
   */
   public        void  setAdresse (AdresseIp adrReseau)
   {
    adrByte1 = adrReseau.getAdresseByte1();
    adrByte2 = adrReseau.getAdresseByte2();
    adrByte3 = adrReseau.getAdresseByte3();
    adrByte4 = adrReseau.getAdresseByte4();
   }

  /**
   * Configure cette instance à la même adresse réseau que l'instance passée en paramètre
   * et configure le numéro de station à 'newAdrStation'
   * Est utilisée pour définir l'adresse d'une interface connectée à une connexion possédant son adresse réseau
   */
   public        void  setAdresse (AdresseIp adrReseau, int  newAdrStation)
   {
    adrByte1 = adrReseau.getAdresseByte1();
    adrByte2 = adrReseau.getAdresseByte2();
    adrByte3 = adrReseau.getAdresseByte3();
    adrByte4 = newAdrStation;
   }

  /**
   * Obtention du byte 'X' de l'adresse Ip de cette instance
   */
   public        int  getAdresseByte1 () { return  adrByte1; }
   public        int  getAdresseByte2 () { return  adrByte2; }
   public        int  getAdresseByte3 () { return  adrByte3; }
   public        int  getAdresseByte4 () { return  adrByte4; }

  /**
   * retourne 'true' si cette instance est une adresse multicast sinon 'False'
   */
   public        boolean  IsMulticastAdresse ()
   {
    if  ( adrByte1 > 223 && adrByte1 < 240 )
    {
      return  true ;
    }
    else
    {
      return  false  ;
    }
   }

  /**
   * retourne 'true' si cette instance à la même adresse que l'instance passée
   * en paramètre sinon 'False'
   */
   public        boolean  compareTo (AdresseIp toAdresse)
   {
    if  ( toAdresse.getAdresseByte1() == adrByte1
        && toAdresse.getAdresseByte2() == adrByte2
        && toAdresse.getAdresseByte3() == adrByte3
        && toAdresse.getAdresseByte4() == adrByte4)
    {
      return  true ;
    }
    else
    {
```

```
      return   false  ;
    }
   }

  /**
   * retourne 'true' si cette instance à la même adresse réseau que
   * l'instance passée en paramètre sinon 'False'
   */
   public          boolean   compareToLan (AdresseIp toAdresse)
   {
    if  ( toAdresse.getAdresseByte1() == adrByte1
        && toAdresse.getAdresseByte2() == adrByte2
        && toAdresse.getAdresseByte3() == adrByte3)
    {
      return   true ;
    }
    else
    {
      return   false  ;
    }
   }

  /**
   * retourne 'true' si cette instance à une adresse réseau inférieure à
   * l'instance passée en paramètre sinon 'False'
   */
   public          boolean   isLower (AdresseIp toAdresse)
   {
    if  ( adrByte1 < toAdresse.getAdresseByte1() )
    {
      return   true ;
    }
    else
    {
      if  ( adrByte1 == toAdresse.getAdresseByte1() )
      {
        if  ( adrByte2 < toAdresse.getAdresseByte2() )
        {
          return   true ;
        }
        else
        {
          if  ( adrByte2 == toAdresse.getAdresseByte2() )
          {
            if  ( adrByte3 < toAdresse.getAdresseByte3() )
            {
              return   true ;
            }
            else
            {
              if  ( adrByte3 == toAdresse.getAdresseByte3() )
              {
                if  ( adrByte4 < toAdresse.getAdresseByte4() )
                {
                  return   true ;
                }
              }
            }
          }
        }
      }
    }

    return   false  ;
   }


/**                                                                **
```

```
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                          */

  /**
   * Affichage de cette adresse IP
   */
  public   String toString ()
  {
    return   new String (    new Integer( getAdresseByte1() ) + "."
                           + new Integer( getAdresseByte2() ) + "."
                           + new Integer( getAdresseByte3() ) + "."
                           + new Integer( getAdresseByte4() )
                         );
  }


/**                                                                        **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                          */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private    void  writeObject(ObjectOutputStream s) throws  IOException
  {
    s.writeInt(this .adrByte1);
    s.writeInt(this .adrByte2);
    s.writeInt(this .adrByte3);
    s.writeInt(this .adrByte4);


    s.writeInt(this .nextAdrMulticastByte1);
    s.writeInt(this .nextAdrMulticastByte2);
    s.writeInt(this .nextAdrMulticastByte3);

    s.writeInt(this .nextAdrReseauByte1);
    s.writeInt(this .nextAdrReseauByte2);
    s.writeInt(this .nextAdrReseauByte3);
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private    void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
  {
    this .adrByte1 = s.readInt();
    this .adrByte2 = s.readInt();
    this .adrByte3 = s.readInt();
    this .adrByte4 = s.readInt();


    this .nextAdrMulticastByte1 = s.readInt();
    this .nextAdrMulticastByte2 = s.readInt();
    this .nextAdrMulticastByte3 = s.readInt();

    this .nextAdrReseauByte1 = s.readInt();
    this .nextAdrReseauByte2 = s.readInt();
    this .nextAdrReseauByte3 = s.readInt();

  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  simulateurpimsm.domaine.message.*;

/**
 * Abstraction représentant une connexion du domaine
 */
abstract  public  class  Connexion extends  ElementDomaine
{
   // adresse réseau de la connexion
   protected   AdresseIp adrConnexion = null ;

   // utilisé pour connaître le dernier num. attribué à une interface
   protected   int  numInterface;

   /**
    * Constructeurs :
    */
   public        Connexion ()
   {
    // définition de l'adresse sous réseau de cette connexion
    adrConnexion = new AdresseIp();
    adrConnexion.setNextAdresseReseau();

    numInterface = 0;
    maxInterfaces = 2;
   }

  /**
   * Entrée d'un message dans cette objet connexion via l'interface 'interf'
   */
   public        void  in (Message msg, Interface interf)
   {
    // Expédition du message sur toutes les interface de cette connexion à l'exception de l'interface 'interf'
    outExept(msg, interf);
   }

  /**
   * Expédition d'un message sur l'interface 'interf'
   */
   public       void  out (Message msg, Interface interf)
   {
    // Expédie le message en entrée sur l'interface
    interf.in(msg);
   }

  /**
   * Gestion de la sortie d'un message via toutes les interfaces de l'élément
   */
  public        void  out (Message msg)
  {
    int  i = 1;
    int  j = this .getNbrInterfaces();

    // traiter toutes les interface de la liste
    while  ( i < j )
    {
      Interface interfTraitee = getInterface(i);
      if  ( interfTraitee !=  null )
      {
        // Expédie le message en entrée sur l'interface traitée
        interfTraitee.in(msg);
      }

      // passage à l'interface suivante de la liste
      i = i + 1;
    }
  }

  /**
   * Expédition d'un message sur toutes les interfaces de cette connexion
   * à l'exception de 'interf'
   */
   public       void  outExept (Message msg, Interface interf)
   {
    int  i = 0;
    int  j = this .getNbrInterfaces();
    // traiter toutes les interface de la liste
    while  ( i < j )
    {
      Interface interfTraitee = getInterface(i);
      if  (interfTraitee != null )
      {
        // Expédie le message en entrée sur l'interface traitée si celle-ci n'est pas 'interf'
        if  ( interfTraitee != interf )
        {
          interfTraitee.in(msg);
        }
      }

      // passage à l'interface suivante de la liste
      i = i + 1;
    }
  }

  /**
   * Ajoute newInterface à la liste des interfaces et configure son adresse Ip
   */
   public        boolean  addInterface (Interface newInterface)
   {
    // appel à la méthode de la classe supérieure pour effectuer l'ajout à la liste des interfaces
    if  ( super .addInterface(newInterface) )
    {
        // Si l'interface est ajoutée, configure son adresse Ip
        AdresseIp adr = new AdresseIp();
        numInterface = numInterface + 1;
        adr.setAdresse(adrConnexion, numInterface);
        newInterface.setAdresse(adr);

        newInterface.setConnexion(this );
        return   true ;
    }
    return   false  ;
   }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;
import  java.io.*;

/**
 * Classe représentant le domaine.
 * Une instance de celle-ci contient tous les éléments du domaine représenté
 * et s'occupe de la configuration de ceux-ci.
 * (Ajout, suppression, Liaison de deux éléments, gestion des tables de routage,
 *  gestion du BSR, des RP et des groupes actifs)
 */
public    class   Domaine implements   Serializable
{
   // Définition des poids respectifs aux différents types de Connexions
   static    final    int  P_LAN = 1;
   static    final    int  P_PAP = 2;

   /**
    * Matrice permettant le calcul du routage du domaine
    */
   private   MatriceRoutage mRoutage = null ;

   /**
    * Vecteur utilisé lors du calcul des routes à partir d'un élément de la matrice de routage
    */
   // Vecteur des interfaces menant de l'élément traité aux autres éléments
   private      Vector interfaceSortie = null ;
   // Vecteur parallèle au précédent indiquant la distance entre les deux
   // éléments en passant par l'interface définie dans le vecteur précédent
   private      Vector distanceTo = null  ;

   /**
    * Vecteur contenant la définition des groupes multicast actifs dans ce domaine
    */
   private   Vector groupeActif  = new Vector();

   /**
    * Liste des éléments actifs contenus dans ce domaine
    */
   private   Vector elementsActifsContenus  = new Vector();

   /**
    * Liste des connexions contenues dans ce domaine
    */
   private   Vector connexionsContenues  = new Vector();

   /**
    * Routeur oeuvrant comme BSR pour ce domaine
    */
   private   Routeur bsr = null ;


   // Constructeurs :
   // _____________
   public   Domaine()
   {
    // Initialisation des différents vecteurs
    groupeActif = new Vector();
    elementsActifsContenus = new Vector();
    connexionsContenues = new Vector();
   }

   /**
    * Election du routeur du domaine oeuvrant comme BSR pour le domaine
    * (C'est le premier Routeur de la liste des éléments Actifs du domaine étant CBSR)
    */
   private         void  setBsr ()
   {
      bsr = null ;

      // parcourir la liste des éléments actifs du domaine
      int  i = 0 ;
      int  j = elementsActifsContenus.size();
      while  (i < j )
      {
        // sélectionné l'élément à traiter
        ElementActif ea = (ElementActif) elementsActifsContenus.elementAt(i);

        // si l'élément traité est un routeur
        if  ( ea instanceof    Routeur)
        {
          Routeur eaRouteur = (Routeur) ea;
          // s'il est CBSR
          if  (eaRouteur.isCBSR())
          {
            // Si aucun BSR sélectionné
            if  (bsr == null )
            {
              // sélectionner celui-ci
              bsr = eaRouteur;
            }
            else
            {
              // si celui-ci posséde une adresseIp plus petite que le BSR sélectionné
              if  ( eaRouteur.getLowerIp().isLower(bsr.getLowerIp()) )
              {
                // sélectionner celui-ci
                bsr = eaRouteur;
              }
            }
          }
        }

        // passer à l'élément suivant
        i = i + 1;
      }
   }

   /**
    * obtention du BSR du domaine
    */
   public         Routeur getBsr ()
   {
    return  bsr;
   }

   /**
    * Election des routeurs du domaine oeuvrant comme RP pour chacun des groupes
    * multicast actifs dans ce domaine
    * ( Les groupes sont partagés équitablement entre les CRP du domaine.
    * Cependant, aucune configuration du choix des groupes possibles par routeur n'est effectuée)
    */
   private         void  setRp ()
   {
    // Boolean utisé pour éviter de boucler s'il n'y a pas de CRP
    boolean   crpExiste = false  ;

    // Variable permettant le parcours des éléments actifs du domaine
    int  indiceEAD = 0;
```

```
    int totalEAD = elementsActifsContenus.size();
    ElementActif ead = null ;

    // Traiter tous les groupes
    int indiceGA = 0;
    int totalGA = groupeActif.size();
    GroupeMulticast ga = null ;

    while (true )
    {
      // Si tous les groupes sont traités, fin de traitement
      if (indiceGA == totalGA)
      {
        return ;
      }

      // Si tous les éléments actifs ont été parcourus
      if (indiceEAD == totalEAD)
      {
        if (crpExiste == false ) // Aucun CRP n'a été trouvé
        {
          return ;
        }
        // revenir au premier élément de la liste
        indiceEAD = 0;
      }

      if ( ga == null )
      {
        // Sélectionner le groupe suivant
        ga = (GroupeMulticast) groupeActif.elementAt(indiceGA);
      }

      // sélectionner l'élément suivant
      ead = (ElementActif) elementsActifsContenus.elementAt(indiceEAD);
      indiceEAD = indiceEAD + 1;

      // si l'élément Actif sélectionné est un routeur
      if ( ead instanceof Routeur)
      {
        Routeur eadRouteur = (Routeur) ead;

        // s'il est CRP
        if ( eadRouteur.isCRP() )
        {
          // le définir comme RP du groupe traité
          ga.setRp(eadRouteur);

          // au moins un CRP existe dans le domaine
          crpExiste = true ;

          // réinitialiser groupe traité
          ga = null ;
          indiceGA = indiceGA + 1;
        }
      }
    }

  }

  /**
   * Obtention du Rp du domaine d'un groupe multicast
   * retourne null si le groupe n'est pas actif sur ce domaine
   * OU que son RP n'est pas attribué.
   */
  public Routeur getRp (AdresseIp adrGroupe)
  {
    // Recherche d'un groupe actif portant l'adresse demandée
    int i = 0;
    int j = groupeActif.size();
    while (i < j )
    {
      // Obtenir le groupe à traiter
      GroupeMulticast gm = (GroupeMulticast) groupeActif.elementAt(i);

      // vérification de l'adresse IP du groupe
      if (gm.getAdresse().compareTo(adrGroupe) == true )
      {
        return gm.getRp();
      }
      // passer au groupe suivant
      i = i + 1;
    }

    // le groupe n'a pas été trouvé
    return null ;
  }

  /**
   * l'élément du domaine fourni en paramètre est-il le Rp d'un Groupe ?
   */
  public boolean isRp(Routeur routeur)
  {
    // Parcours de la liste des groupes actifs
    int i = 0;
    int j = groupeActif.size();
    while (i < j )
    {
      // Obtenir le groupe à traiter
      GroupeMulticast gm = (GroupeMulticast) groupeActif.elementAt(i);

      // comparaison du RP du groupe et du routeur demanadé
      if (gm.getRp() == routeur)
      {
        return true ;
      }

      // passer au groupe suivant
      i = i + 1;
    }

    // le routeur n'a pas été trouvé
    return false ;
  }

  /**
   * effectue le calcul des routes entre les éléments du domaine
   * et configure les tables de routage de chacun de ces éléments.
   */
  public void route ()
  {
    // Initialisation de la matrice de routage
    prepareMatriceRoutage();

    // Traiter tous les éléments du domaine compris dans la matrice de routage
    int x = 0;
    int y = mRoutage.getNbrElement();
    while ( x < y )
    {
      // intialiser le traitement de l'élément
      initRoutage(x);

      // Calculer les routes partant de celui-ci
      executeDijkstra(x);

      // Sauvegarder le résultat du calcul directement dans l'élément
      clotureRoutage(x);

      // Passer à l'élément suivant de la matrice
```

```
      x = x + 1;
    }
  }

  /**
   * Création de la matrice de routage et de chaque ligne de celle-ci
   * ( Ne configure pas la ligne de routage, ne fait que la créer !!!)
   */
  private        void  prepareMatriceRoutage ()
  {
    // initialiser la matrice de routage de ce domaine
    mRoutage = new MatriceRoutage();

    // Traiter les Eléments Actifs du domaine
    //---------------------------------------
    int  x = 0;
    int  y = elementsActifsContenus.size();
    while  ( x < y )
    {
      // obtenir l'élément actif
      ElementActif ea = (ElementActif) elementsActifsContenus.elementAt(x);

      // Si l'élément Actif est un routeur, l'ajouter à la la matrice de routage
      if  ( ea instanceof     Routeur)
      {
       mRoutage.addElement(ea);
      }
      // Passer à l'élément suivant
      x = x +1;
    }

    // Traiter les Connexions du domaine
    // ---------------------------------
    x = 0;
    y = connexionsContenues.size();
    while  ( x < y )
    {
      // obtenir la connexion
      Connexion con = (Connexion)connexionsContenues.elementAt(x);

      // Si la connexion est un Lan, l'ajouter à la la matrice de routage
      if  ( con instanceof     Lan)
      {
       mRoutage.addElement(con);
      }
      // Passer à l'élément suivant
      x = x +1;
    }

    // Initialisation avant traitement des interfaces de chaque élément de la matrice
    // ------------------------------------------------------------------------------
    mRoutage.initInterfaces();

    // Configuration de la liste des interfaces de chaque ligne de la matrice
    // ----------------------------------------------------------------------
    x = 0;
    y = mRoutage.getNbrElement();
    while  ( x < y )
    {
      // Obtention de l'élément à traiter
      ElementDomaine ed = mRoutage.getElement(x);

      // traiter la liste des interfaces de cet objet
      int  x1 = 0;
      int  y1 = ed.getNbrInterfaces();
      while  ( x1 < y1 )
      {
        // Obtenir l'interface
        Interface interf = ed.getInterface(x1);
        if  (interf != null )
        {
          // Obtenir l'objet du domaine à l'autre extrémité de l'interface traitée
          ElementDomaine edExtrem = null ;
          int  poids = this .P_LAN;
          if  ( ed instanceof     Connexion)
          {
            edExtrem = interf.getElementActif();
          }
          else  // C'est un routeur
          {
            edExtrem = interf.getConnexion();

            // Si ce routeur est connecté à une ligne PàP (non géré pour le routage)
            if  ( edExtrem instanceof     PaP)
            {
              PaP edExtremPaP = (PaP) edExtrem;

              // sélectionner le routeur situé à l'autre extrémité de cette ligne PàP
              edExtrem = edExtremPaP.linkedTo(interf);

              // le poids de cette connexion est de 2
              poids = this .P_PAP;
            }
          }

          // si l'extrémité existe
          if  (edExtrem != null )
          {
            // Si l'autre extrémité est un objet de la matrice de routage, Ajouter l'interface
            Integer i = mRoutage.getX(edExtrem);
            if  (i != null )
            {
              mRoutage.setInterface(x, i.intValue(), interf);
              mRoutage.setPoids(x, i.intValue(), poids);
            }
          }
        }

        // Traiter l'interface suivante de la ligne
        x1 = x1 + 1 ;
      }

      // Passer à la ligne suivante.
      x = x + 1;
    }

  }

  /**
   * Initialisation du calcul de routage pour l'élément traité de la matrice
   */
  private        void  initRoutage (int  ofX)
  {
    // Initialisation de l'élément du domaine
    ElementDomaine ed = mRoutage.getElement(ofX);
    if  ( ed instanceof     Routeur )
    {
      Routeur edRouteur = (Routeur) ed;
      edRouteur.getUnicast().init();
    }
    else
    {
      Lan edLan = (Lan) ed;
      edLan.initAssert();
    }

    // Initialisation des attributs du domaine jouant un rôle dans le calcul des routes
    interfaceSortie = new Vector();
    interfaceSortie.setSize(mRoutage.getNbrElement());
```

```
    distanceTo = new Vector();
    distanceTo.setSize(mRoutage.getNbrElement());

    //l'élément ofX est la racine (Lui donner une distance égale à 0)
    distanceTo.setElementAt(new Integer(0), ofX);

    // Initialiser les First Hop et l'interface menant à ceux-ci
    int i = 0;
    int j = mRoutage.getNbrElement();
    while (i < j)
    {
      Interface interf = mRoutage.getInterface(ofX, i);

      // si l'élément traité est connecté à la racine
      if (interf != null )
      {
        // configuré la distance et l'interface de sortie
        distanceTo.setElementAt(new Integer(mRoutage.getPoids(ofX, i)), i);
        interfaceSortie.setElementAt(interf, i);
      }

      // Elément suivant
      i = i + 1;
    }
  }

  /**
   * Exécution de l'algorithme de Dijkstra pour le calcul des routes à partir de l'élément
   * du domaine d'indice 'root' dans la matrice de routage précédement initialisé.
   * (Cest à dire avec tous les chemins menant aux routeurs directement voisins de la racine configurés.)
   */
  private          void  executeDijkstra (int  root)
  {
    // entrer au moins une fois dans la boucle
    boolean   amelioration Trouvee = true ;

    // nombre d'itérations déjà effectuées
    int  nbrIter = 1;

    // effectuer le traitement tant que des améliorations sont trouvées
    while  (amelioration Trouvee)
    {
      // initialisé le traitement
      amelioration Trouvee = false  ;

      // Traiter tous les éléments
      int  i = 0;
      int  j = mRoutage.getNbrElement();
      while  (i < j )
      {
        Integer dist = (Integer) distanceTo.elementAt(i);

        // si l'élément traité est déjà accessible à partir de la racine OU est la racine
        if  ( dist != null )
        {
            // Si la distance entre la racine et cet élément est < au nombre d'itération,
            // Alors il n'y a plus d'amélioration possible à partir de cet élément
            if  ( dist.intValue() >= nbrIter )
            {
              // traiter tous les éléments
              int  iTo = 0;
              int  jTo = mRoutage.getNbrElement();
              while  ( iTo < jTo )
              {
                // obtenir l'interface entre les deux éléments
                Interface interf = mRoutage.getInterface(i, iTo);

                // si cette interface existe
                if  ( interf != null  )
                {
                  // la distance de la racine à l'élément 'iTo' en passant par l'élément 'i'
                  int  newDt = dist.intValue() + mRoutage.getPoids(i, iTo);

                  // Si une améliorable est possible
                  boolean   ameliorable = false  ;
                  Integer dt = (Integer) distanceTo.elementAt(iTo);
                  if  (dt == null  )
                  {
                    ameliorable = true ;
                  }
                  else
                  {
                    if  ( dt.intValue() > newDt )
                    {
                      ameliorable = true ;
                    }
                  }

                  if  ( ameliorable )
                  {
                    //  une amélioration est trouvée
                    amelioration Trouvee = true ;

                    // la distance entre la racine et l'élément d'indice 'iTo' = 'distanceTraitee'+ poid de la ligne
                    distanceTo.setElementAt(new Integer(newDt), iTo);

                    // l'interface de sortie à partir de la racine vers l'élément d'indice 'iTo'
                    // est la même que l'interface menant à l'élément d'indice 'i'
                    interfaceSortie.setElementAt(interfaceSortie.elementAt(i), iTo );
                  }
                }

                // passer à l'élément suivant
                iTo = iTo + 1;
              }
            }
        }

        // passer à l'élément suivant
        i = i + 1;
      }

      // une itération supplémentaire à été effectuée
      nbrIter = nbrIter + 1;
    }
  }

  /**
   * Clôture du calcul des routes de l'élément en cours de traitement
   * ( redéfinition des tables de routage de l'élément traité )
   */
  private          void  clotureRoutage (int  ofX)
  {
   // Obtention de l'élément racine traité
   ElementDomaine ed = mRoutage.getElement(ofX);

   int  x = 0;
   int  y = mRoutage.getNbrElement();
   while  ( x < y )
   {
     // Traiter tous les éléments sauf l'élement d'indice 'ofX'
     if  ( x != ofX)
     {
       // obtention de l'interface traitée
       Interface interf = (Interface) interfaceSortie.elementAt(x);

       // Si interf est différent de null, l'élément d'indice x est accessible à partir
       // de l'élément d'indice 'ofX' via l'interface 'interf'
       if  ( interf != null  )
```

```
      {
        if  ( ed instanceof    Routeur )
        {
          Routeur edRouteur = (Routeur) ed;
          edRouteur.getUnicast().setRoute(mRoutage.getElement(x) , interf);
        }
        else  // ed est un Lan
        {
          Lan edLan = (Lan) ed;
          edLan.setAssert(mRoutage.getElement(x) , interf);
        }
      }
    }
    // passer à l'élément suivant
    x = x + 1;
  }
}

/**
 * Méthode:       addElement
 * Retour :       /
 * Description:  Ajout d'un élément au domaine
 */
public        void  addElement (Connexion newElement)
{
 connexionsContenues.addElement(newElement);
 newElement.setDomaine(this );
}

public        void  addElement (ElementActif newElement)
{
 elementsActifsContenus.addElement(newElement);
 newElement.setDomaine(this );
}

/**
 * Méthode:       removeElement
 * Retour :       /
 * Description:  Suppression d'un élément du domaine
 */
public        void  removeElement (ElementDomaine element)
{
 // Déconnexion de toutes les interfaces de l'élément
 int  i = 0;
 int  j = element.getNbrInterfaces();

 // traiter toutes les interfaces
 while  (i < j)
 {
   // obtention de l'interface à traiter
   Interface interf = null ;
   interf = element.getInterface(i);

   // Déconnexion de l'autre extrémité de l'interface
   if  (element instanceof    ElementActif)
   {
     Lan lan = null ;

     // Si l'interface relie un routeur à un Lan
     if  ( (interf.getConnexion() instanceof    Lan)
        && (interf.getElementActif() instanceof    Routeur) )
     {
       // si ce routeur est actuellement le DR de ce Lan
       if  ( interf.isDr() )
       {
         // Aprés déconnexion, Il sera nécessaire de recalculer un DR pour ce Lan
         lan = (Lan) interf.getConnexion();
       }
     }

     // déconnecter l'interface de la connexion
     interf.getConnexion().removeInterface(interf);

     // recalcul du Dr du Lan si besoin
     if  (lan != null )
     {
       lan.setDr();
     }
   }
   else
   {
     interf.getElementActif().removeInterface(interf);
   }

   // Passer à l'interface suivante
   i = i + 1;
 }

 // Suppression de l'élément
 if  (element instanceof    ElementActif)
 {
   elementsActifsContenus.removeElement(element);
 }
 else
 {
   connexionsContenues.removeElement(element);
 }


 // Si l'élément à supprimer est un routeur
 if  (element instanceof    Routeur)
 {
   Routeur elementRouteur = (Routeur) element;

   // recalculer le BSR si besoin
   if  ( elementRouteur == getBsr() ) { setBsr(); }

   // recalculer les RP si besoin
   if  ( isRp(elementRouteur) )  { setRp(); }
 }

 // recalcul des routes si besoin
 if  ( !(element instanceof    Station) ) { route(); }
}

/**
 * Connexion d'un élément actif du domaine à une connexion du domaine
 *
 * Si Ok retourne true sinon false
 */
public        boolean  connectElements (ElementActif fromElementActif, Connexion toConnexion)
{
 // Création d'une nouvelle interface
 Interface interf = new Interface();

 // Ajout de cette interface à la connexion
 // (ET configuration de l'adresse IP de cette interface par la connexion)
 if  ( toConnexion.addInterface(interf))
 {
   // Ajout de cette interface (adresse IP configurée) à l'élément Actif
   if  ( fromElementActif.addInterface(interf) )
   {
     // La connexion à toConnexion est établie
     //=> Si l'élément actif est un Routeur
     if  (fromElementActif instanceof    Routeur)
     {
       if  ( toConnexion instanceof    Lan )
       {
           Lan lan = (Lan) toConnexion;
           lan.setDr();
```

```
        }

        // Recalcul du routage dans le domaine si l'élément actif est un routeur
        this .route();

        // Réinitialisation de PIM SM (les interfaces des TIB ne sont plus à jour)
        this .initPimSm();
      }

      // La connexion est établie
      return  true ;
    }
    else  // La connexion à l'élément actif n'est pas établie
    {
      // suppression de l'interface ajoutée à la connexion
      toConnexion.removeInterface(interf);
    }
  }

  // La connexion n'est pas établie
  return  false ;
 }

 /**
  * Connexion de deux éléments actifs du domaine
  * ( via une ligne PàP )
  *
  * Si Ok retourne true sinon false
  */
 public        boolean  connectElements (Routeur fromRouteur, Routeur toRouteur)
 {
  // Création des nouveaux objets
  Interface interf1 = new Interface();
  Interface interf2 = new Interface();
  PaP pap = new PaP();

  // Ajout des interfaces à la connexion PaP
  // ET configuration des adresses IP
  pap.addInterface(interf1);
  pap.addInterface(interf2);

  // Ajout des interfaces (adresse IP configurée) aux routeurs
  if  ( fromRouteur.addInterface(interf1) )
  {
     if ( toRouteur.addInterface(interf2) )
     {
        // la connexion est établie
        return  true ;
     }
     else
     {
        // si la connexion à toRouteur à échoué, supprimer la connexion à fromRouteur
        fromRouteur.removeInterface(interf1);
     }
  }

  // la connexion n'est pas établie
  return  false ;
 }

 /**
  * Suppression d'une connexion entre deux éléments du domaine
  *
  * Si Ok retourne true sinon false
  */
 public        boolean  deconnectElements (ElementActif fromElement, Connexion toConnexion)
 {
  // recherche de l'interface connectant les deux éléments
  Interface interf = fromElement.getInterfaceTo(toConnexion);

  // Si l'interface existe, supprimer la connexion des deux côtés
  if  ( interf != null  )
  {
      interf.getConnexion().removeInterface(interf);

      fromElement.removeInterface(interf);

      if  (fromElement instanceof    Routeur)
      {
        // Recalcul du routage dans le domaine
        this .route();

        // Réinitialise PIM SM (les interfaces des TIB ne sont plus à jour)
        this .initPimSm();
      }

      return  true ;
  }

  // la déconnexion n'est pas possible
  return  false ;
 }

 /**
  * Configure un routeur en tant que Candidat BSR ou non.
  * Si besoin, recalcul le BSR du domaine
  */
 public        void  setCBSR (Routeur routeur, boolean  etat)
 {
    // Si le routeur est dans la liste des éléments actifs du domaine
    if  ( elementsActifsContenus.contains(routeur) )
    {
      // si l'état doit être changé
      if  (routeur.isCBSR() != etat)
      {
          // modifier l'état
          routeur.setCBSR(etat);

          // recalculer le BSR du domaine
          this .setBsr();
      }
    }
 }

 /**
  * Configure un routeur en tant que Candidat RP ou non.
  * Si besoin, recalcul les RP des groupes actifs du domaine
  */
 public        void  setCRP (Routeur routeur, boolean  etat)
 {
    // Si le routeur est dans la liste des éléments actifs du domaine
    if  ( elementsActifsContenus.contains(routeur) )
    {
      // si l'état doit être changé
      if  (routeur.isCRP() != etat)
      {
          // modifier l'état
          routeur.setCRP(etat);

          // recalculer les RP du domaine
          this .setRp();
      }
    }
 }

 /**
  * Configure une station du domaine pour recevoir le flux multicast issu du groupe portant l'adresse Ip 'adrGroupe'
  */
 public        void  addRecever (Station station, AdresseIp adrGroupe)
```

```
  {
    // Vérifie que la station appartient au domaine
    if  ( elementsActifsContenus.contains(station) )
    {
      // Vérifie que le groupe multicast utilisant l'adresse 'adrGroupe' est bien actif sur ce domaine
      if  (isGroupeActif(adrGroupe))
      {
        // Configure la station
        station.addRecever(getGroupeActif(adrGroupe));
      }
    }
  }

  /**
   * Configure une station du domaine pour recevoir le flux multicast issu d'une station source précise à
   * destination du groupe portant l'adresse Ip 'adrGroupe'
   */
  public        void  addRecever (Station station, Station source, AdresseIp adrGroupe)
  {
    // Vérifie que la station appartient au domaine
    if  ( elementsActifsContenus.contains(station) )
    {
      // Vérifie que la source appartient au domaine
      if  ( elementsActifsContenus.contains(source) )
      {
        // Vérifie que le groupe multicast utilisant l'adresse 'adrGroupe' est bien actif sur ce domaine
        if  (isGroupeActif(adrGroupe))
        {
          // Configure la station
          station.addRecever(source, getGroupeActif(adrGroupe));
        }
      }
    }
  }

  /**
   * suppression de la réception du flux d'un groupe multicast utilisant l'adresse 'ardGroupe'
   * sur une station du domaine
   */
  public        void  removeReceveur (Station station, AdresseIp adrGroupe)
  {
    // Vérifie que la station appartient au domaine
    if  ( elementsActifsContenus.contains(station) )
    {
      // Vérifie que le groupe multicast utilisant l'adresse 'adrGroupe' est bien actif sur ce domaine
      if  (isGroupeActif(adrGroupe))
      {
        // Configure la station
        station.removeRecever(getGroupeActif(adrGroupe));
      }
    }
  }

  /**
   * suppression de la réception du flux issu d'une station source précise à destination
   * d'un groupe multicast utilisant l'adresse 'ardGroupe' sur une station du domaine
   */
  public        void  removeReceveur (Station station, Station source, AdresseIp adrGroupe)
  {
    // Vérifie que la station appartient au domaine
    if  ( elementsActifsContenus.contains(station) )
    {
      // Vérifie que la source appartient au domaine
      if  ( elementsActifsContenus.contains(source) )
      {
        // Vérifie que le groupe multicast utilisant l'adresse 'adrGroupe' est bien actif sur ce domaine
        if  (isGroupeActif(adrGroupe))
        {
          // Configure la station
          station.removeRecever(source, getGroupeActif(adrGroupe));
        }
      }
    }
  }

  /**
   * Configuration d'une station du domaine pour émettre un flux multicast en direction du groupe portant l'adresse Ip
'adrGroupe'
   */
  public        void  addSource (Station source, AdresseIp adrGroupe)
  {
    // Vérifie que la source appartient au domaine
    if  ( elementsActifsContenus.contains(source) )
    {
      // Vérifie que le groupe multicast utilisant l'adresse 'adrGroupe' est bien actif sur ce domaine
      if  (isGroupeActif(adrGroupe))
      {
        // Configure la station
        source.addSource(getGroupeActif(adrGroupe));
      }
    }
  }

  /**
   * Configuration d'une station du domaine pour ne plus émettre un flux multicast en direction du groupe portant l'adresse
Ip 'adrGroupe'
   */
  public        void  removeSource (Station source, AdresseIp adrGroupe)
  {
    // Vérifie que la source appartient au domaine
    if  ( elementsActifsContenus.contains(source) )
    {
      // Vérifie que le groupe multicast utilisant l'adresse 'adrGroupe' est bien actif sur ce domaine
      if  (isGroupeActif(adrGroupe))
      {
        // Configure la station
        source.removeSource(getGroupeActif(adrGroupe));
      }
    }
  }

  /**
   * Activation d'un nouveau groupe multicast dans ce domaine
   * Si l'adresse demandée correspond à l'adresse d'un groupe multicast déjà actif rien n'est changé.
   */
  public        void  activeGroupeMulticast(AdresseIp newAdrMulticast)
  {
    boolean   trouve = false  ;

    // Recherche d'un groupe actif portant l'adresse demandée
    int  i = 0;
    int  j = groupeActif.size();
    while  (i < j )
    {
      // Obtenir le groupe à traiter
      GroupeMulticast gm = (GroupeMulticast) groupeActif.elementAt(i);

      // vérification de l'adresse IP du groupe
      if  (gm.getAdresse().compareTo(newAdrMulticast) == true  )
      {
        trouve = true ;
        break ;
      }
      // passer au groupe suivant
      i = i + 1;
    }

    // si le groupe n'existe pas
    if  (trouve == false  )
```

```
    {
      // création du groupe
      GroupeMulticast gm = new GroupeMulticast ();
      gm.setAdresse(newAdrMulticast);

      // Ajout du groupe à la liste des groupes actifs
      groupeActif.addElement(gm);

      // Lui affecter un RP
      this .setRp();
    }
  }

  /**
   * Désactive le groupe multicast utilisant l'adresse IP 'adrMulticast' sur ce domaine
   */
  public     void  desactiveGroupeMulticast (AdresseIp adrMulticast)
  {
    // Recherche d'un groupe actif portant l'adresse demandée
    int  i = 0;
    int  j = groupeActif.size();
    while  (i < j )
    {
      // Obtenir le groupe à traiter
      GroupeMulticast gm = (GroupeMulticast) groupeActif.elementAt(i);

      // vérification de l'adresse IP du groupe
      if  (gm.getAdresse().compareTo(adrMulticast) == true  )
      {
        // suppression du groupe dans la liste des groupes actifs
        groupeActif.removeElementAt(i);
        break ;
      }
      // passer au groupe suivant
      i = i + 1;
    }
  }

  /**
   * Un groupe multicast utilisant l'adresse IP 'adrMulticast' est)il actif sur ce domaine ?
   */
  public     boolean  isGroupeActif (AdresseIp adrMulticast)
  {
    GroupeMulticast gm = getGroupeActif(adrMulticast);
    if  ( gm == null )
    {
      return  false ;
    }

    return  true ;
  }

  /**
   * retourne le groupe multicast actif dans ce domaine utilisant l'adresse 'adrMulticast'
   * ( s'il n'existe pas, retourne null )
   */
  public     GroupeMulticast getGroupeActif (AdresseIp adrMulticast)
  {
    // Recherche d'un groupe actif portant l'adresse demandée
    int  i = 0;
    int  j = groupeActif.size();
    while  (i < j )
    {
      // Obtenir le groupe à traiter
      GroupeMulticast gm = (GroupeMulticast) groupeActif.elementAt(i);

      // vérification de l'adresse IP du groupe
      if  (gm.getAdresse().compareTo(adrMulticast) == true  )
      {
        return  gm;
      }
      // passer au groupe suivant
      i = i + 1;
    }

    // le groupe n'a pas été trouvé
    return  null ;
  }

  /**
   * Obtention du nombre de connexions
   */
  public     int  getNbrConnexion()
  {
    return  this .connexionsContenues.size();
  }

  /**
   * Obtention de la connexion d'indice 'indice'
   * (Si 'indice' est trop grand, return null)
   */
  public     Connexion getConnexion (int  indice)
  {
    if  ( indice > connexionsContenues.size())
    {
      return  null ;
    }

    return  (Connexion) connexionsContenues.elementAt(indice);
  }

  /**
   * Obtention du nombre d'éléments actifs
   */
  public     int  getNbrElementActif()
  {
    return  this .elementsActifsContenus.size();
  }

  /**
   * Obtention de l'éléments actifs d'indice 'indice'
   * (Si 'indice' est trop grand, return null)
   */
  public     ElementActif getElementActif (int  indice)
  {
    if  ( indice > elementsActifsContenus.size())
    {
      return  null ;
    }

    return  (ElementActif) elementsActifsContenus.elementAt(indice);
  }

  /**
   * Obtention du nombre de groupe actif
   */
  public     int  getNbrGroupeActif()
  {
    return  groupeActif.size();
  }

  /**
   * Obtention du groupe actif d'indice 'indice'
   * (Si 'indice' est trop grand, return null)
   */
  public     GroupeMulticast getGroupeActif (int  indice)
  {
    if  ( indice > groupeActif.size())
    {
```

```
        return  null ;
      }

      return  (GroupeMulticast) groupeActif.elementAt(indice);
  }

  /**
   * Initialisation du protocol PimSm sur ce domaine
   */
  public  void  initPimSm()
  {
    // initialisation de chaque Lan du domaine
    int  i = 0;
    int  j = this .connexionsContenues.size();
    while  (i < j)
    {
      // Obtention de l'élément actif
      Connexion c = (Connexion) this .connexionsContenues.elementAt(i);

      if  (c != null )
      {
        // Si l'élément actif traité est un LAN
        if  (c instanceof  Lan)
        {
          Lan cLan = (Lan) c;

          // Ré-initialisation du protocol PimSm de ce routeur
          cLan.initPimSm();
        }
      }

      // traiter la connexion
      i = i + 1;
    }

    // initialisation PimSm de chaque routeur du domaine
    i = 0;
    j = elementsActifsContenus.size();
    while  (i < j)
    {
      // Obtention de l'élément actif
      ElementActif ea = (ElementActif) elementsActifsContenus.elementAt(i);

      if  (ea != null )
      {
        // Si l'élément actif traité est un routeur
        if  (ea instanceof  Routeur)
        {
          Routeur eaRouteur = (Routeur) ea;

          // Ré-initialisation du protocol PimSm de ce routeur
          eaRouteur.initPimSm();
        }
      }

      // traiter l'élément actif suivant
      i = i + 1;
    }

    // Initialisation de chaque receveur et émmetteur du domaine
    i = 0;
    while  (i < j)
    {
      // Obtention de l'élément actif
      ElementActif ea = (ElementActif) elementsActifsContenus.elementAt(i);

      if  (ea != null )
      {
        // Si l'élément actif traité est un routeur
        if  (ea instanceof  Station)
        {
          Station eaStation = (Station) ea;

          // Ré-initialisation du protocol PimSm de cette station
          eaStation.initPimSm();
        }
      }

      // traiter l'élément actif suivant
      i = i + 1;
    }
  }


  /**                                                                    **
   * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
   **                                                                    */

    /**
     * Affichage de cette adresse IP
     */
    public  String toString ()
    {
      return  new String ("domaine multicast");
    }

    /**
     * Liste des groupes Multicast actifs
     */
    public  String toStringLstGroupeMulticast ()
    {
      String text = new String();

      int  i = 0;
      int  j = getNbrGroupeActif();
      while  ( i < j )
      {
        GroupeMulticast gm = getGroupeActif(i);
        if  (gm != null )
        {
          String rp = new String("NON DEFINI");
          if  (gm.getRp() != null  )
          {
            rp = "" + gm.getRp();
          }

          text = text + "groupe actif num. " + new Integer(i+1) + "\n"
                      + "-->  Adr IP : " + gm.getAdresse() + "\n"
                      + "-->  RP : " + rp + "\n"
                      + "-->  Couleur : " + gm.getColor() + "\n\n";
        }
        i = i +1;
      }
      return  text;
    }

    /**
     * Liste des Elements du domaine
     */
    public  String toStringLstElements ()
    {
      String sbsr = new String("NON DEFINI");
      if  (bsr != null )
      {
        sbsr = "" + this .bsr;
      }

      String text = this .toString() + " [ BSR :" + sbsr + "]\n\n";
      text = text + "1) Elements Actifs : ";
```

```
    text = text + "\n____________________\n";

    int  i = 0;
    int  j = this .elementsActifsContenus.size();
    while  ( i < j )
    {
      ElementActif ea = (ElementActif) this .elementsActifsContenus.elementAt(i);
      if  (ea != null )
      {
        String element = ea.toStringConfig();
        if  (element != null )
        {
          text = text + element + "\n";
        }
      }
      i = i +1;
    }

    text = text + "\n2) Connexions :";
    text = text + "\n_________________\n";

    i = 0;
    j = this .connexionsContenues.size();
    while  ( i < j )
    {
      Connexion cn = (Connexion) this .connexionsContenues.elementAt(i);
      if  (cn != null )
      {
        String element = cn.toStringConfig();
        if  (element != null )
        {
          text = text + element + "\n";
        }
      }
      i = i +1;
    }

    return  text;
  }


/**                                                                    **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                    */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private  void  writeObject(ObjectOutputStream s) throws  IOException
  {
    // Sauvegarde des éléments Actifs
    s.writeObject(this .elementsActifsContenus);

    // Sauvegarde des connexions
    s.writeObject(this .connexionsContenues);

    // Sauvegarde des groupes actifs
    s.writeObject(this .groupeActif);
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private  void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
  {
    // Recupération des éléments actifs
    this .elementsActifsContenus = (Vector) s.readObject();

    // Recupération des connexions
    this .connexionsContenues = (Vector) s.readObject();

    // Recupération des groupes actifs
    this .groupeActif = (Vector) s.readObject();

    // Recherche du BSR
    this .setBsr();

    // Recherche des RP
    this .setRp();

    // recalcul des routes
    this .route();

    // initialisation du protocol Pim SM
    this .initPimSm();

  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  simulateurpimsm.domaine.message.*;


/**
 * Abstraction représentant un élément Actif du domaine
 */
abstract  public  class  ElementActif extends  ElementDomaine
{
   // Variables permettant l'attribution automatique d'un num aux interfaces de cet élément
   protected      int  nextNumInterf = 0;

   // Constructeurs :
   // _____________
   public         ElementActif ()
   {
    maxInterfaces = 2;
   }

  /**
   * Gestion de la sortie d'un message via toutes les interfaces de l'élément
   */
   public         void  out (Message msg)
   {
    int  i = 1;
    int  j = this .getNbrInterfaces();

    // traiter toutes les interfaces de la liste
    while  ( i < j )
    {
      Interface interfTraitee = getInterface(i);

      if  ( interfTraitee != null )
      {
        // Expédie le message en sortie sur l'interface traitée
        interfTraitee.out(msg);
      }

      // passage à l'interface suivante de la liste
      i = i + 1;
    }
   }

  /**
   * Expédition d'un message sur l'interface 'interf'
   */
   public        void  out (Message msg, Interface interf)
   {
    if  (interf != null )
    {
      // Expédie le message en sortie sur l'interface
      interf.out(msg);
    }
   }

  /**
   * Expédition d'un message sur toutes les interfaces de cette connexion
   * à l'exception de 'interf'
   */
   public        void  outExept (Message msg, Interface interf)
   {
    int  i = 1;
    int  j = this .getNbrInterfaces();

    // traiter toutes les interface de la liste
    while  ( true  )
    {
      Interface interfTraitee = getInterface(i);

      if  ( interfTraitee != null  )
      {
        // Expédie le message sur l'interface traitée si celle-ci n'est pas 'interf'
        if  ( interfTraitee != interf )
        {
          interfTraitee.out(msg);
        }
      }

      // passage à l'interface suivante de la liste
      i = i + 1;
    }
   }

  /**
   * retourne l'interface directement connectée à la
   * connexion 'toConnexion' appartenant à la liste des interfaces
   * de cette instance si elle existe sinon null
   */
   public        Interface getInterfaceTo (Connexion toConnexion)
   {

    int  i = 0;
    int  j = this .getNbrInterfaces();

    // parcourir la liste des interfaces de cette instance
    while  ( i < j )
    {
        Interface interf = getInterface(i);

        if  ( interf != null  )
        {
          // si l'interface est connectée à la connexion demandée, la retourner
          if  ( interf.getConnexion() == toConnexion )
          {
           return  interf;
          }
        }

        // passer à l'interface suivante de la liste de cette instance
        i = i + 1;
    }

    // L'interface n'a pas été trouvée
    return  null ;
   }

  /**
   * Ajoute newInterface à la liste des interfaces
   * et conecte cet élément actif à cette nouvelle interface
   *
   * Retour :      true si l'interface à bien été ajoutée
   *               false si le nombre max. d'interface est déja atteint
   */
   public        boolean  addInterface (Interface newInterface)
   {
    // appel à la méthode de la classe supérieure pour effectuer l'ajout à la liste des interfaces
    if  (super .addInterface(newInterface) == false  )
    {
      return  false  ;
```

```
    }

    newInterface.setElementActif(this );

    // Configuration du num. de cette interface
    nextNumInterf = nextNumInterf + 1;
    newInterface.numInterf = nextNumInterf;

    return  true ;
   }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;
import  simulateurpimsm.domaine.message.*;

/**
 * Abstraction représentant un élément du domaine
 * et servant de base aux objets Connexion et Elément Actif
 */
abstract  public  class  ElementDomaine
{
   // Nombre maximum d'interfaces intégrées dans ce type d'élément
   protected  int  maxInterfaces;

   // Vecteur contenant les interfaces de l'élément
   protected  Vector interfaceContenue = null ;

   // Domaine auquel appartient cet élément Actif.
   protected  Domaine domaine = null ;

  /**
   * Constructeur:
   */
  public  ElementDomaine()
  {
    // Initialisation de la liste d'interface
    interfaceContenue = new Vector();
    maxInterfaces = 0;
  }

  /**
   * Configuration du domaine auquel appartient cet élément?
   */
  public  void  setDomaine (Domaine newDomaine)
  {
    domaine = newDomaine;
  }

  /**
   * Obtention du domaine auquel appartient cet élément
   */
  public  Domaine getDomaine()
  {
    return  domaine;
  }


  /**
   * retourne le nombre d'interface réellement intégrée dans cette instance
   */
   public  int  getNbrInterfaces ()
   {
    return  interfaceContenue.size();
   }

  /**
   * Retourne le nombre maximum d'interfaces pouvant être intégrées dans cette instance
   */
   public  int  getMaxInterfaces ()
   {
    return  maxInterfaces;
   }

  /**
   * Gestion de l'entrée d'un message via l'une des interfaces de l'élément
   */
   abstract  public  void  in (Message msg, Interface interf);

  /**
   * Gestion de la sortie d'un message via toutes les interfaces de l'élément
   */
  abstract  public  void  out (Message msg);

  /**
   * Si le nombre max. d'interface de cet élément n'est pas déja atteint, ajoute
   * 'newInterface' à cette instance. return true
   * Sinon return false
   */
  public  boolean  addInterface (Interface newInterface)
  {
    // On ne peut pas dépasser le nombre maximum d'interfaces
    if  ( getNbrInterfaces() >= maxInterfaces )
    {
     return  false  ;
    }

    interfaceContenue.addElement(newInterface);
    return  true  ;
  }

  /**
   * Retourne l'interface d'indince 'indice' contenue dans la liste d'interface de l'instance
   * OU null si celle-ci est inexistante ou si l'indice est trop grand.
   */
   public  Interface getInterface (int  indice)
   {
    // L'idice doit être plus petit ou  égal au nombre maximum d'interfaces
    if  (indice >= this .getNbrInterfaces())
    {
      return  null  ;
    }

    return  (Interface) interfaceContenue.get(indice);
   }

  /**
   * Supprime l'interface 'oldInterface' de la liste d'inetreface de l'instance
   * si celle-ci existe. Sinon, retourne une Exception PimSm
   *
   * Retour :       'true' si l'interface à été suprimée de la liste de cette instance
   *                'false' si l'interface n'a pas été trouvée dans la liste
   */
   public  boolean  removeInterface (Interface oldInterface)
   {
    return  interfaceContenue.removeElement(oldInterface);
   }


/**                                                                    **
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                     */


   /**
    * Affichage du nom de ce routeur
    */
   public  String toString ()
   {
      return  new String("Element Domaine");
   }
```

```
    /**
     * Configuration détaillée de ce routeur
     */
    public  String  toStringConfig ()
    {
        return  this .toString();
    }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package simulateurpimsm.domaine;

import java.io.*;

/**
 * Définition d'un flux multicast :
 * Le flux peut-être du type (*,G) si la source n'est pas définie
 * Ou (S,G) si la source est définie
 */
public class Fluxmulticast implements Serializable
{
  /**
   * Source spécifique de ce flux
   */
  protected Station source = null ;

  /**
   * Groupe Multicast auquel appartient ce flux
   */
  protected GroupeMulticast groupe = null ;

  /**
   * Configuration de la station émettrice de ce flux
   */
  public void setSource (Station newSource)
  {
   source = newSource;
  }

  /**
   * Obtention de la station émettrice de ce flux
   */
  public Station getSource ()
  {
   return source;
  }

  /**
   * Configuration du groupe multicast de ce flux
   */
  public void setGroupe (GroupeMulticast newGroupe)
  {
   groupe = newGroupe;
  }

  /**
   * Obtention du groupe multicast de ce flux
   */
  public GroupeMulticast getGroupe ()
  {
   return groupe;
  }



 /**                                                                **
  * --------  Méthodes utilisées pour l'interfacage utilisateur ----------- *
  **                                                                */

 public String toString ()
  {
    String res = new String ( "flux(");

    if ( this .source == null )
    {
      res = res + "*";
    }
    else
    {
      res = res + this .source;
    }

    res = res + ", " + this .groupe + ")";

    return res;
  }


/**                                                                **
 * --------  Méthodes utilisée pour la sérialisation ---------------------- *
 *                                                                */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private void writeObject(ObjectOutputStream s) throws IOException
  {
    // Sauvegarde de la source de ce flux
    s.writeObject(this .source);

    // Sauvegarde du groupe de ce flux
    s.writeObject(this .groupe);
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private void readObject(ObjectInputStream s) throws IOException, ClassNotFoundException
  {
    // Recupération de l'adresse
    this .source = (Station) s.readObject();
    this .groupe = (GroupeMulticast) s.readObject();
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.awt.Color;
import  java.io.*;

/**
 * Classe s'occupant de la gestion d'un groupe multicast
 * (et plus particulièrement, de son adresse IP et du RP de celui-ci)
 */
public    class   GroupeMulticast implements   Serializable
{
  /**
   * Routeur du domaine oeuvrant comme RP pour ce groupe
   */
   private      Routeur rp = null ;

   /**
    * Adresse IP multicast du groupe
    */
   private      AdresseIp adresseGroupe = null ;

   /**
    * Configuration de l'adresse IP du groupe
    */
   public        void  setAdresse (AdresseIp newAdr)
   {
    adresseGroupe = newAdr;
   }

   /**
    * Obtention de l'adresse Ip du groupe
    */
   public        AdresseIp getAdresse ()
   {
    return   adresseGroupe;
   }

   /**
    * Configuration du RP du groupe
    */
   public        void  setRp (Routeur newRp)
   {
    rp = newRp;
   }

   /**
    * Obtention du RP du groupe
    */
   public        Routeur getRp ()
   {
    return   rp;
   }



  /**                                                                  **
   * --------  Méthodes utilisées pour l'interfacage utilisateur ----------- *
   **                                                                  */

    // Couleur représentant ce groupe Multicast dans le simulateur.
    Color    color = null ;

    public   String toString ()
    {
      return   "" + this .getAdresse();
    }

    /**
     * Définition de la couleur
     */
     public      void  setColor(Color newColor)
     {
      color = newColor;
     }

    /**
     * Obtention de la couleur
     */
     public      Color getColor()
     {
        return   color;
     }


/**                                                                  **
 * --------  Méthodes utilisée pour la sérialisation ---------------------- *
 *                                                                  */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private    void  writeObject(ObjectOutputStream s) throws   IOException
  {
    // Sauvegarde de l'adresse
    s.writeObject(this .adresseGroupe);
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private    void  readObject(ObjectInputStream s) throws   IOException, ClassNotFoundException
  {
    // Recupération de l'adresse
    this .adresseGroupe = (AdresseIp) s.readObject();
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package simulateurpimsm.domaine;

import java.util.Vector;

/**
 * Objet contenant toutes les informations propres à une interface pour un
 * flux multicast (*,G) spécifique de la TIb d'un routeur Pim SM
 */
public class GrpIntTIB
{
   private boolean igmp = false ;
   private boolean join = false ;

   private Interface interf = null ;

   /**
    * Configuration de l'interface désignée par cette entrée
    */
   public void setInterface(Interface newInterface)
   {
      interf = newInterface;
   }

   /**
    * Obtention de l'interface désignée par cette entrée
    */
   public Interface getInterface()
   {
      return interf;
   }

   /**
    * Définit l'arrivée d'un Message IGMP Join Ou prune par cette interface en provenance
    * d'un LAN pour lequel ce routeur est DR
    */
   public void setIgmp (boolean etat)
   {
      igmp = etat;
   }

   /**
    * Un message IGMP join est-il arrivé par cette interface en provenance d'un
    * LAN pour lequel ce routeur est DR
    */
   public boolean isIgmp ()
   {
      return igmp;
   }

   /**
    * Défini l'arrivée d'un Join par cette interface en provenance d'un routeur
    */
   public void setJoin (boolean etat)
   {
      join = etat;
   }

   /**
    * Un Join est-il arrivé en provenance d'un autre routeur par cette interface
    */
   public boolean isJoin ()
   {
      return join;
   }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;

/**
 * Objet contenant toutes les informations de la TIb d'un routeur Pim SM
 * concernant un groupe multicast
 * ( c'est-à-dire flux (*,G) ET flux (S,G) !!! )
 */
public    class  GrpTIB
{
   /**
    * Adresse IP du groupe
    */
   private     AdresseIp adrGroupe = null ;

   /**
    * Mode de gestion de ce groupe pour le DR
    * ( passé en mode Shortest Path Tree ou non )
    */
   private     boolean  modeSPT = false ;

   private   Vector lstSource  = null ;
   private   Vector lstInterface  = null ;

   /**
    * Constructeur
    */
   public   GrpTIB()
   {
      init();
   }

   /**
    * Initialisation du groupe
    */
   public   void  init()
   {
      lstSource  = new Vector();
      lstInterface  = new Vector();
   }

   /**
    * Configuration de l'adresse Ip de ce groupe
    */
   public        void  setAdrGroupe (AdresseIp newAdrGroupe)
   {
      adrGroupe = newAdrGroupe;
   }

   /**
    * Obtention de l'adresse Ip de ce groupe
    */
   public        AdresseIp getAdrGroupe ()
   {
      return  adrGroupe;
   }

   /**
    * Ajout d'une interface à la liste des interfaces par lesquelles
    * le flux de ce groupe doit-être transféré
    * ( Si l'interface est déja présente, ne fait rien )
    */
   public        GrpIntTIB setInterface (Interface newInterface)
   {
      // Recherche de cette interface dans la liste des interfaces
      int  i = 0;
      int  j = lstInterface.size();

      // traiter toute la liste
      while  (i < j)
      {
        // Obtenir l'interface traitée
        GrpIntTIB git = (GrpIntTIB) lstInterface.elementAt(i);

        if  ( git != null  )
        {
          // si l'interface traitée est la même que newInterface ne pas continuer
          if  ( git.getInterface() == newInterface )
          {
            return  git;
          }
        }

        // Traiter l'interface suivante
        i = i + 1;
      }

      // l'interface n'existe pas => l'ajouter
      GrpIntTIB git = new GrpIntTIB();
      git.setInterface(newInterface);
      lstInterface.addElement(git);
      return  git;
   }


   /**
    * suppression d'une interface de la liste des interfaces par lesquelles
    * le flux de ce groupe doit-être transféré
    */
   public        void  removeInterface (Interface interf)
   {
      // Recherche de cette interface dans la liste des interfaces
      int  i = 0;
      int  j = lstInterface.size();

      // traité toute la liste
      while  (i < j)
      {
        // Obtenir l'interface traitée
        GrpIntTIB git = (GrpIntTIB) lstInterface.elementAt(i);

        if  ( git != null )
        {
          // si l'interface traitée est la même que 'interf'
          if  ( git.getInterface() == interf )
          {
            lstInterface.removeElementAt(i);
            return  ;
          }
        }

        // Traiter l'interface suivante
        i = i + 1;
      }

   }

   /**
    * Obtention du nombre d'interface par lesquelles le flux
```

```
   * de ce groupe doit-être transféré
   */
  public        int  getNbrInterface ()
  {
      return  lstInterface.size();
  }

  /**
   * Obtention du détail pour ce flux de l'interface d'indice 'indice'
   * ( Retourne null si l'indice est trop grand )
   */
  public        GrpIntTIB getInterface (int  indice)
  {
     // si l'indice est trop grand, retourne null
     if  ( indice > getNbrInterface())
     {
       return  null ;
     }

     return  (GrpIntTIB) lstInterface.elementAt(indice);
  }

  /**
   * Obtention du détail pour ce flux de l'interface 'interf'
   * ( Retourne null si l'interface n'est pas trouvée )
   */
  public        GrpIntTIB getInterface (Interface interf)
  {
     // traiter toutes les interfaces
     int  i = 0;
     int  j = getNbrInterface();
     while  (i < j)
     {
       // Obtention du détail interface traitée
       GrpIntTIB git = (GrpIntTIB) lstInterface.elementAt(i);

       if  ( git != null )
       {
         // si l'objet est celui recherché, le retourner
         if  (git.getInterface() == interf)
         {
           return  git;
         }
       }

       // traiter l'interface suivante
       i = i + 1;
     }

     // l'interface n'a pas été trouvée
     return  null ;
  }

  public        void  setModeSpt (boolean  etat)
  {
     modeSPT = etat;
  }

  public        boolean  isModeSPT ()
  {
     return  modeSPT;
  }
  /**
   * Obtention du nombre de flux propres à une source pour ce groupe
   */
  public        int  getNbrInfoSource ()
  {
      return  lstSource.size();
  }

  /**
   * Obtention du détail d'une source propre à ce flux d'indice 'indice'
   * ( Retourne null si l'indice est trop grand )
   */
  public        SrcGrpTIB getInfoSource (int  indice)
  {
     // si l'indice est trop grand, retourne null
     if  ( indice > getNbrInfoSource())
     {
       return  null ;
     }

     return  (SrcGrpTIB) lstSource.elementAt(indice);
  }

  /**
   * Obtention du détail d'une source propre à ce flux
   * ( Retourne null si pas trouvée )
   */
  public        SrcGrpTIB getInfoSource (ElementActif source)
  {
     // traiter toutes les sources de ce groupe
     int  i = 0;
     int  j = lstSource.size();
     while  (i < j)
     {
       // Obtention du détail de la source traitée
       SrcGrpTIB sgt = (SrcGrpTIB) lstSource.elementAt(i);

       if  (sgt != null )
       {
         // si l'objet est celui recherché, le retourner
         if  (sgt.getSource() == source)
         {
           return  sgt;
         }
       }

       // traiter la source suivante
       i = i + 1;
     }

     // la source n'a pas été trouvée
     return  null ;
  }

  /**
   * Ajout d'un flux (S,G) propre à une source pour ce groupe
   * et retour de celui-ci.
   * (Si le flux est déjà présent, ne le modifie pas)
   */
  public        SrcGrpTIB newInfoSource (Station newSource)
  {
     // Ne créer l'instance que si elle n'existe pas déjà
     SrcGrpTIB sgt = getInfoSource(newSource);
     if  (sgt == null )
     {
       // créé l'instance de l'objet
       sgt = new SrcGrpTIB();
       sgt.setSource(newSource);

       // L'ajouter à la liste
       lstSource.addElement(sgt);
     }

     return  sgt;
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  simulateurpimsm.domaine.message.*;
import  java.io.*;

/**
 * Classe représentant une interface connectant un élément actif à une connexion
 */
public    class   Interface implements   Serializable
{
   // Connexion à laquelle l'interface est connectée
   private      Connexion connexion = null ;

   // Elément actif auquel l'interface appartient
   private      ElementActif elementActif = null ;

   // Si l'élément actif est un routeur et que la connexion est un Lan, en est-il le DR
   private      boolean   isDr = false  ;

   // Adresse IP de l'interface (défiie par la connexion
   private      AdresseIp adrInterface = null ;

   // Num attribué à cette interface par l'élément actif lors de sa connexion
   protected       int   numInterf = 0;


  /**
   * Défini l'adresse Ip de l'interface
   */
   public         void  setAdresse (AdresseIp newAdresse)
   {
    adrInterface = newAdresse;
   }

  /**
   * obtention de l'adresse Ip de l'interface
   */
   public         AdresseIp getAdresse ()
   {
    return    adrInterface;
   }

  /**
   * Entrée d'un message vers l'élément actif
   */
   public         void  in (Message msg)
   {
    if  (elementActif != null )
    {
      elementActif.in(msg, this );
    }
   }

  /**
   * Sortie d'un message vers la connexion
   */
   public         void  out (Message msg)
   {
    if  (connexion != null )
    {
      connexion.in(msg, this );
    }
   }

   /**
   * Connecte l'interface à la connexion 'newConnexion'
   */
   public         void  setConnexion (Connexion newConnexion)
   {
    connexion = newConnexion;
   }

  /**
   * Retourne la connexion à laquelle l'interface est connectée
   */
   public         Connexion getConnexion ()
   {
    return   connexion;
   }

   /**
   * Connecte l'interface à l' élément Actif 'newElementActif'
   */
   public         void  setElementActif (ElementActif newElementActif)
   {
    elementActif = newElementActif;
   }

  /**
   * retourne l'élément actif auqel appartient l'interface
   */
   public         ElementActif getElementActif ()
   {
    return   elementActif;
   }

   /**
    * Configuration de isDr
    *
    * Rem : Est utilisée par un Lan pour définir le Dr de celui-ci
    */
    public         void  setDr(boolean   etat)
    {
      isDr = etat;
    }

   /**
    * Cette interface est-elle connectée au DR d'un Lan
    */
    public         boolean   isDr()
    {
      return   isDr;
    }


/**                                                                              **
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                             */


   /**
    * Affichage du nom de cette interface
    */
   public   String toString ()
   {
      return   new String("I." + new Integer(this .numInterf));
   }

   /**
    * Configuration détaillée de ce Lan
```

```
    */
   public   String  toStringConfig ()
   {
      String  adr = new String("NON DEFINI");
      if  (this .adrInterface != null )
      {
        adr = "" + this .adrInterface;
      }

      return   this .toString() + "[ AdrIp:" + adr + " ]";
   }


/**                                                                    **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                      */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private   void  writeObject(ObjectOutputStream s) throws  IOException
  {
    // Sauvegarde du num. d'interface
    s.writeInt(this .numInterf);

    // Sauvegarde de l'adresse
    s.writeObject(this .adrInterface);

    // Sauvegarde des éléments du domaine
    s.writeObject(this .connexion);
    s.writeObject(this .elementActif);

    // Sauvegarde de l'état DR
    s.writeBoolean(this .isDr);

  }

  /**
   * Restauration de l'instance de l'objet
   */
  private   void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
  {
    // Recupération du num. d'interface
    this .numInterf = s.readInt();

    // Recupération de l'adresse
    this .adrInterface = (AdresseIp) s.readObject();

    // Recupération des éléments du domaine
    this .connexion = (Connexion) s.readObject();
    this .elementActif = (ElementActif) s.readObject();

    // Recupération de l'état DR
    this .isDr = s.readBoolean();

  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;
import  simulateurpimsm.domaine.message.*;
import  java.io.*;

/**
 * Classe représentant une connexion Lan du domaine
 */
public    class   Lan extends   Connexion implements   Serializable
{
   // Interface menant directement au DR de ce Lan
   protected      Interface DrWinner = null ;

   protected      Vector lstAssert  = new Vector();

   // Constructeurs :
   // _____________
   public         Lan ()
   {
    maxInterfaces = 20;
   }

   /**
    * Gestion d'une demande IGMP pour un 'flux'
    * en provenance d'une station connectée à 'interfEntree'
    *
    * Si la demande est une fin de réception, le Lan interroge toutes les autres
    * stations afin de déterminer si le flux n'est plus demandé par personne.
    * Si c'est le cas, transmet la demande au DR. Sinon, n'en tient pas compte.
    *
    * Rem : etat = true     pour une demande de réception
    *       etat = false    pour une demande de fin de réception
    */
   public         void  igmp (Fluxmulticast flux,boolean   etat, Interface interfEntree)
   {
      int  i = 0;
      int  j = this .getNbrInterfaces();
      Interface interfToDr = null ;

      // Parcours de toutes les interfaces interfaces
      while  ( i < j )
      {
        // Obtenir l'interface traitée
        Interface interfTraitee = this .getInterface(i);

        if  (interfTraitee != null )
        {
          // Si l'interface traitée est celle menant au Dr, la retenir
          if  ( interfTraitee.isDr() )
          {
             interfToDr = interfTraitee;
          }

          // Si la demande est une fin de réception
          if  ( etat == false   )
          {
            // Si l'interface est connectée à une station
            if  ( interfTraitee.getElementActif() instanceof    Station )
            {
              // Si cette station n'est pas l'émettrice de la demande
              if  ( interfTraitee != interfEntree)
              {
                Station station = (Station) interfTraitee.getElementActif();

                // Si cette station désire toujours recevoir ce flux
                if  ( station.getFlux(Station.RECUS, flux.getGroupe(), flux.getSource()) != null   )
                {
                  // fin de traitement (le flux n'est pas supprimé sur ce LAN )
                  return  ;
                }
              }
            }
          }

          // passer à l'interface suivante
          i = i + 1;
        }
      }


      // Si l'interface menant au DR est trouvée (Normalement toujours)
      if  ( interfToDr != null   )
      {
        // passer la demande au DR
        Routeur routeurDr = (Routeur) interfToDr.getElementActif();
        routeurDr.igmp(flux, interfToDr, etat);
      }
   }

   /**
    * Initialisation
    */
   public         void  initAssert ()
   {
      lstAssert = new Vector();
   }

   /**
   * Initilisation PimSm ( Configuration du DR de ce Lan )
   */
   public      void  initPimSm()
   {
     this .setDr();
   }

   /**
   * Ajout d'une nouvelle ligne de routage à la table
   * ou Remplacement de la ligne pointant sur newElement
   * si celle-ci est déja existante
   */
   public         void  setAssert (ElementDomaine newElement, Interface newInterface)
   {
      // recherche d'une ligne de routage dans la table pointant déja sur cet élément
      LigneRoutage ligne = getLigneRoutage (newElement);

      // Si cette ligne de routage existe déja, la supprimer
      if  ( ligne != null   )
      {
        lstAssert.removeElement(ligne);
      }

      // Création de la ligne de routage
      ligne = new LigneRoutage(newElement, newInterface);

      // Ajout de cette nouvelle ligne de routage à la table
      lstAssert.addElement(ligne);
   }

  /**
```

```
 * Description:  retourne l'interface de sortie menant à un element du domaine
 *
 * Retour : Interface associée à elementDomaine
 *          s'il existe une ligne dans la table de routage pointant sur celui-ci
 *          sinon null
 */
public        Interface getAssert (ElementDomaine elementDomaine)
{
 // recherche d'une ligne de routage dans la table pointant déja sur cet élément
 if  (elementDomaine instanceof     Station)
 {
   // Si l'élément est une station, prendre comme destination la connexion attachée à celle-ci
   Station st = (Station) elementDomaine;
   elementDomaine = st.getConnexion();
 }
 LigneRoutage ligne = getLigneRoutage (elementDomaine);

 // Si cette ligne de routage existe déja, la supprimer
 if  ( ligne != null  )
 {
   return   ligne.getInterface();
 }

 // aucune ligne n'a été trouvée
 return   null ;
}

/**
* recherche d'une ligne pointant sur un élément du domaine dans la table de routage
*
* Retour :       LigneRoutage pointant sur l'élement demandé ou null si cette ligne n'existe pas dans cette table
*/
private       LigneRoutage getLigneRoutage(ElementDomaine element)
{
   LigneRoutage ligne = null ;
   int  i = 0;

   // traiter toutes les lignes de la table
   int  to = lstAssert.size();
   while  (i < to)
   {
     // obtention de la ligne traitée
     ligne = (LigneRoutage) lstAssert.get(i);

     // si l'élément pointé dans la ligne traitée est celui demandé, retourner cette ligne
     if  (ligne.getElement() == element )
     {
       return   ligne;
     }

     //passer à l'élément suivant
     i = i + 1;
   }

   // aucune ligne de routage ne pointe sur l'élément recherché.
   return   null ;
}

/**
*  Intérrogation des routeurs connecté à ce LAN affin de vérifier que l'
*  émission d'un flux en direction des autres routeurs n'est plus
*  nécessaire sur ce LAN.
*  Cette méthode est utilisée par le DR afin de simuler le timer
*  permettant de temporiser la prise en compte d'un Prune et ce,
*  dans le but de laisser le temps à un autre routeur de contrer ce Prune.
*
*  Rem:  - flux = description (S,G) Ou (*,G)
*        - rpt  = true et flux (S,G) => PruneRpt(S,G)
*        - interfIn = Interface connectée au routeur effectuant la demande (normallement DR)
*/
public        boolean  VerificationPruneOk (Fluxmulticast flux, boolean  rpt, Interface interfIn)
{
   int  i = 0;
   int  j = this .getNbrInterfaces();

   // Traiter toutes les interfaces
   while  ( i < j )
   {
     // Obtenir l'interface traitée
     Interface interfTraitee = this .getInterface(i);

     if  ( interfTraitee != null )
     {
       // si l'interface n'est pas l'interface d'entrée de la demande
       if  ( interfTraitee != interfIn)
       {
         // si l'interface traitée est connectée à un routeur
         if  ( interfTraitee.getElementActif() instanceof     Routeur)
         {
           Routeur routeurTraite = (Routeur) interfTraitee.getElementActif();
           // Si le routeur traité à toujours besoin de ce flux, Stop
           if  (routeurTraite.isFluxNecessaire(flux, rpt, interfTraitee) == true )
           {
             return   false  ;
           }
         }
       }
     }

     // passer à l'interface suivante
     i = i + 1;
   }

   // aucun routeur n'a plus besoin du flux traité
   return   true ;
}

/**
 *  Election du DR pour ce Lan
 */
public       void  setDr()
{
   int  i = 0;
   int  j = this .getNbrInterfaces();

   // réinitialise DrWinner
   if  ( DrWinner != null )
   {
     DrWinner.setDr(false  );
     DrWinner = null ;
   }

   // Traiter toutes les interfaces
   while  ( i < j )
   {
     // Obtenir l'interface traitée
     Interface interfTraitee = this .getInterface(i);

     if  ( interfTraitee != null )
     {
       // si l'interface traitée est connectée à un routeur
       if  ( interfTraitee.getElementActif() instanceof     Routeur)
       {
         // S'il n'y a actuellement pas de Dr
         if  ( DrWinner == null  )
         {
           // Le routeur connecté à cet interfac est le DR momentané.
           DrWinner = interfTraitee;
         }
```

```
            else
            {
              // Si l'adresse Ip de cette interface est inférieure à l'adresse du DR
             if  ( interfTraitee.getAdresse().isLower(DrWinner.getAdresse()) )
             {
                DrWinner = interfTraitee;
             }
            }
          }
        }

        // passer à l'interface suivante
        i = i + 1;
      }

      if  ( DrWinner != null )
      {
        DrWinner.setDr(true );
      }
  }

  /**
   * Entrée d'un message dans cette objet connexion via l'interface 'interf'
   */
   public        void  in (Message msg, Interface interf)
   {
      // si le message est un message register, register Stop, d'un Prune ou d'un Join
      if  ( (msg instanceof     Register) || (msg instanceof     RegisterStop) ||
            (msg instanceof     JoinG) || (msg instanceof     PruneG) || (msg instanceof     JP_SG) )
      {
        // Obtenir l'interface menant à la destination
        Interface interfOut = this .getAssert( msg.getDestination() );
        if  ( interfOut != null )
        {
          // sortir ce message uniquement par cette interface
          interfOut.in(msg);
        }
      }
      else
      {
          // passer par la gestion normale des connexions
          super .in(msg, interf);
      }
   }

  /**                                                                    **
   * --------  Méthodes utilisées pour l'interfacage utilisateur ---------- *
   **                                                                     */

  public  String toString ()
  {
    return  new String ( "Lan." + adrConnexion.getAdresseByte3() );
  }

   /**
    * Configuration détaillée de ce Lan
    */
   public  String  toStringConfig ()
   {
      String res = this .toString() ;
      res = res + " [ Adr IP="
                + new Integer(this .adrConnexion.getAdresseByte1() ) + "."
                + new Integer(this .adrConnexion.getAdresseByte2() ) + "."
                + new Integer(this .adrConnexion.getAdresseByte3() ) + ".xx"
                + " / " ;

      String rp = new String("NON DEFINI");
      if  ( DrWinner != null )
      {
          rp = "DR=" + DrWinner.getElementActif();
      }
      res = res + rp + " ]\n ";


      int  i = 0;
      int  j = this .getNbrInterfaces();
      if  (j > 0 )
      {
        res = res + "- Interfacé à : [";
        while  (i < j)
        {
          Interface interf = this .getInterface(i);
          String element = new String ("NON DEFINI");
          if  (interf != null )
          {
            if  (interf.getElementActif() != null )
            {
              element = "" + interf.getElementActif();
            }
          }
          res = res + " " + element + " ";

          i = i + 1;
        }
        res = res + " ]\n";
      }
      else
      {
        res = res + "\n";
      }

      return  res;
    }


  /**                                                                    **
   * --------  Méthodes utilisée pour la sérialisation ---------------------- *
   *                                                                      */

    /**
     * Sauvegarde de cette instance de l'objet
     */
    private  void  writeObject(ObjectOutputStream s) throws  IOException
    {
      // Sauvegarde de l'interface menant au DR de ce lan
      s.writeObject(this .DrWinner);

      // Sauvegarde des données propres aux connexions
      s.writeObject(this .adrConnexion);
      s.writeInt(this .numInterface);

      // Sauvegarde des données propres aux Elements du domaine
      s.writeInt(this .maxInterfaces);
      s.writeObject(this .interfaceContenue);
      s.writeObject(this .domaine);
    }

    /**
     * Restauration de l'instance de l'objet
     */
    private  void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
    {
      // Recupération de l'adresse
      this .DrWinner = (Interface) s.readObject();

      // Recupération des données propres aux connexions
      this .adrConnexion = (AdresseIp) s.readObject();
      this .numInterface = s.readInt();
```

```
    // Recupération des données propres aux Elements du domaine
    this .maxInterfaces = s.readInt();
    this .interfaceContenue = (Vector) s.readObject();
    this .domaine = (Domaine) s.readObject();
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package simulateurpimsm.domaine;

import java.util.Vector;

/**
 * Ligne de la matrice de routage concernant un élément du domaine
 * ainsi que toutes les interfaces possibles entre cet élément
 * et les autres éléments de la matrice.
 */
public class LigneMatriceRoutage
{
   /**
    * Elément du domaine concerné par cette ligne de la matrice
    */
   private ElementDomaine element = null ;

   /**
    * Vecteur des interfaces menant directement de l'élément concerné à un autre élément du domaine
    */
   private Vector to  = null ;

   /**
    * configure l'élément concerné par cette ligne de la matrice
    */
   public        void setElement (ElementDomaine newElement)
   {
      element = newElement;
   }

   /**
    * Obtention de l'élément concerné par cette ligne
    */
   public        ElementDomaine getElement ()
   {
      return element;
   }

   /**
    * Initialise le vecteur contenant la liste des interfaces pour contenir 'nbrElement'
    */
   public        void initTo (int nbrElement)
   {
      to = new Vector();

      //Ajouter à ce nouveau vecteur nbrElement vide
      int i = 0;
      while (i < nbrElement)
      {
        // Ajouter un élément ToInteface au vecteur (avec interface = null)
        to.addElement(new ToInterface() );

        // Traiter l'élément suivant
        i = i + 1;
      }
   }

   /**
    * Configuration de l'interface menant à l'élément d'indice 'toX'
    */
   public        void setInterface (int toX, Interface newInterface)
   {
      // Obtention de l'élément
      ToInterface ti = (ToInterface) to.elementAt(toX);

      // configuration de l'interface de cet élément
      ti.setInterface(newInterface);
   }

   /**
    * Configuration du poids de la ligne menant à l'élément d'indice 'toX'
    */
   public        void setPoids (int toX, int poids)
   {
      // Obtention de l'élément
      ToInterface ti = (ToInterface) to.elementAt(toX);

      // configuration de l'interface de cet élément
      ti.setPoids(poids);
   }

   /**
    * Obtention de l'interface menant à l'élément d'indice 'toX'
    */
   public        Interface getInterface (int toX)
   {
      // Obtention de l'élément d'indice 'toX'
      ToInterface ti = (ToInterface) to.elementAt(toX);

      // retour de l'interface de cet élément
      return ti.getInterface();
   }

   /**
    * Obtention du poids du chemin menant à l'élément d'indice 'toX'
    */
   public        int getPoids (int toX)
   {
      // Obtention de l'élément d'indice 'toX'
      ToInterface ti = (ToInterface) to.elementAt(toX);

      // retour de l'interface de cet élément
      return ti.getPoids();
   }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

/**
 * Classe représentant une ligne de la table de routage utilisée pour le routage unicast
 */
public    class  LigneRoutage
{
   // pointeur sur un élément du domaine
   private   ElementDomaine elementRelie = null ;

   // pointeur sur l'interface menant à cet élément
   private   Interface interfaceDeSortie = null ;

   /**
    * Constructeur:
    */
   public        LigneRoutage (ElementDomaine newElement, Interface newInterface)
   {
    elementRelie = newElement;
    interfaceDeSortie = newInterface;
   }

  /**
   * retourne l'élément du domaine pointé par cette ligne de routage
   */
   public        ElementDomaine getElement ()
   {
    return   elementRelie;
   }

  /**
   * retourne l'interface de sortie pointée par cette ligne de routage
   */
   public        Interface getInterface ()
   {
    return   interfaceDeSortie;
   }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;

/**
 * Matrice permettant d'effectuer le calcul des routes entre les éléments du domaine
 */
public    class   MatriceRoutage
{

   // Vecteur contenant chaque ligne de la matrice (une ligne = un élément)
   private   Vector element  = null ;

   /**
    * Constructeur
    */
   public   MatriceRoutage()
   {
    init();
   }

   /**
    * Initialisation de l'objet
    */
   public        void  init ()
   {
    // Initialisation d'un nouveau vecteur d'élément vide
    element  = new Vector();
   }

   /**
    * Retourne le nombre d'élément contenu dans cette matrice de routage
    */
   public        int  getNbrElement ()
   {
      int  res = 0;

      // Si le vecteur est initialisé, retrouver le nombre d'élément de celui-ci
      if  ( element != null )
      {
        res = element.size();
      }

      return   res;
   }

   /**
    * Initialisation de la liste des interfaces des éléments de la matrice
    * est à utilisé aprés avoir ajouté tous les éléments de la matrice
    * et avant de configurer les interfaces de ces éléments.
    */
   public     void  initInterfaces()
   {
      int  i = 0;
      int  j = this .element.size();
      while  (i < j)
      {
        LigneMatriceRoutage lmr = (LigneMatriceRoutage) this .element.elementAt(i);

        if  (lmr != null )
        {
          lmr.initTo(j);
        }

        i = i + 1;
      }
   }


   /**
    * Ajout d'un élément du domaine à la matrice de routage
    */
   public        void  addElement (ElementDomaine newElement)
   {
      // Création d'un nouvelle ligne de routage pour newElement
      LigneMatriceRoutage lmr = new LigneMatriceRoutage();
      lmr.setElement(newElement);

      // Ajout de la nouvelle ligne de routage à cette matrice
      element.addElement(lmr);
   }

   /**
    * Retourne l'indice de la ligne concernant l'élément 'ofElement' dans cette matrice de routage.
    * Si 'ofElement' n'est pas présent dans la matrice, retourne null
    */
   public        Integer getX (ElementDomaine ofElement)
   {
      // Traiter tous les lignes de la matrice
      int  i = 0;
      int  j = element.size();
      while  ( i < j )
      {
        // Obtention de la ligne
        LigneMatriceRoutage lmr = (LigneMatriceRoutage) element.elementAt(i);

        // Si la ligne concerne 'ofElement', retourner un Integer initialisé à i
        if  ( lmr.getElement() == ofElement)
        {
          return   new Integer(i);
        }

        // Passer à la ligne suivante
        i = i +1;
      }

      // La ligne cherchée n'est pas présente.
      return   null ;
   }

   /**
    * Retourne l'élément du domaine concerné par la ligne d'indice 'ofX'
    * Si l'indice est plus grand que la taille de la matrice, retourne null
    */
   public        ElementDomaine getElement (int  ofX)
   {
      // Si l'indice est plus grand que la taille de la matrice, retourne null
      if  (ofX >= element.size() )
      {
        return   null ;
      }

      // Obtention de la ligne
      LigneMatriceRoutage lmr = (LigneMatriceRoutage) element.elementAt(ofX);

      // retour de l'élément concerné par cette ligne
      return   lmr.getElement();
   }

   /**
```

```
   * Configuration de l'interface menant directement de l'élément d'indice 'from' à l'élément d'indice 'to'
   */
  public        void  setInterface (int  from, int    to, Interface interf)
  {
     // obtention de la ligne d'indice 'from'
     LigneMatriceRoutage lmr = (LigneMatriceRoutage) element.elementAt(from);

     if  ( lmr != null  )
     {
       // Ajout de l'interface à la ligne obtenue
       lmr.setInterface(to, interf);
     }
  }

  /**
   * Configuration du poid de la ligne menant directement de l'élément d'indice 'from' à l'élément d'indice 'to'
   */
  public        void  setPoids (int  from, int    to, int  poid)
  {
     // obtention de la ligne d'indice 'from'
     LigneMatriceRoutage lmr = (LigneMatriceRoutage) element.elementAt(from);

     if  ( lmr != null  )
     {
       // Ajout de l'interface à la ligne obtenue
       lmr.setPoids(to, poid);
     }
  }


  /**
   * Obtention de l'interface menant directement de l'élément d'indice 'from' à l'élément d'indice 'to'.
   * Si cette interface n'existe pas, retourne null
   */
  public        Interface getInterface (int  from, int  to)
  {
     // obtention de la ligne d'indice 'from'
     LigneMatriceRoutage lmr = (LigneMatriceRoutage) element.elementAt(from);

     // retourne l'interface de la ligne obtenue menant à l'élément d'indice 'to
     return  lmr.getInterface(to);
  }

  /**
   * Obtention du poids du chemin menant directement de l'élément d'indice 'from' à l'élément d'indice 'to'.
   * Si cette interface n'existe pas, retourne null
   */
  public        int  getPoids (int  from, int  to)
  {
     // obtention de la ligne d'indice 'from'
     LigneMatriceRoutage lmr = (LigneMatriceRoutage) element.elementAt(from);

     // retourne l'interface de la ligne obtenue menant à l'élément d'indice 'to
     return  lmr.getPoids(to);
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;
import  java.io.*;

/**
 * Classe représentant une connexion point à point Actif du domaine
 */
public    class   PaP extends   Connexion implements   Serializable
{
   // Variables statiques permettant l'attribution automatique d'un num aux routeurs
   protected       static    int  nextNumPaP = 0;

   // Num attribué à ce routeur lors de sa création
   protected       int  numPaP = 0;


   /**
   * Constructeur:
   */
   public   PaP()
   {
    // initialisation du numéro de cette ligne PàP
    nextNumPaP = nextNumPaP + 1;
    numPaP = nextNumPaP;
   }

   /**
    * Retourne l'élément actif connecté à l'interface situé à
    * l'autre extrémité de cette ligne PàP que l'interface passée en paramètre
    * (Si elles existent !!)
    */
  public        ElementActif  linkedTo(Interface interf)
  {
      Interface interf1 = this .getInterface(0);
      Interface interf2 = this .getInterface(1);

      if  (interf1 == interf)
      {
        if  (interf2 != null )
        {
          return   interf2.getElementActif();
        }
      }
      else
      {
        if  (interf2 == interf )
        {
          if  (interf1 != null )
          {
            return   interf1.getElementActif();
          }
        }
      }

      return   null ;
  }


/**                                                                          **
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                         */


  /**
   * Affichage de la table de routage unicast
   */
  public   String toString ()
  {
     return   new String("PàP."+ new Integer(this .numPaP) );
  }

  /**
   * Configuration détaillée de cette connexion PaP
   */
  public   String  toStringConfig ()
  {
     String res = this .toString();

     res = res + " [ Adr IP="
               + new Integer(this .adrConnexion.getAdresseByte1() ) + "."
               + new Integer(this .adrConnexion.getAdresseByte2() ) + "."
               + new Integer(this .adrConnexion.getAdresseByte3() ) + ".xx"
               + " / " ;

     String connect1 = new String ("NON DEFINI");
     if  (this .getInterface(0) != null )
     {
       if  (this .getInterface(0).getElementActif() != null )
       {
         connect1 = ""+ this .getInterface(0).getElementActif();
       }
     }

     String connect2 = new String ("NON DEFINI");
     if  (this .getInterface(1) != null )
     {
       if  (this .getInterface(1).getElementActif() != null )
       {
         connect2 = ""+ this .getInterface(1).getElementActif();
       }
     }

     res = res + " relie : " + connect1 + " à " + connect2 + " ]\n";

     return   res;
  }


/**                                                                          **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                          */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private   void  writeObject(ObjectOutputStream s) throws  IOException
  {
    // Sauvegarde des attributs de cette instance PàP
    s.writeInt(this .nextNumPaP);
    s.writeInt(this .numPaP);

    // Sauvegarde des données propres aux connexions
    s.writeObject(this .adrConnexion);
    s.writeInt(this .numInterface);

    // Sauvegarde des données propres aux Elements du domaine
    s.writeInt(this .maxInterfaces);
    s.writeObject(this .interfaceContenue);
    s.writeObject(this .domaine);
```

```
  }

  /**
   * Restauration de l'instance de l'objet
   */
  private   void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
  {
    // Recupération de l'adresse
    this .nextNumPaP = s.readInt();
    this .numPaP = s.readInt();

    // Recupération des données propres aux connexions
    this .adrConnexion = (AdresseIp) s.readObject();
    this .numInterface = s.readInt();

    // Recupération des données propres aux Elements du domaine
    this .maxInterfaces = s.readInt();
    this .interfaceContenue = (Vector) s.readObject();
    this .domaine = (Domaine) s.readObject();
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  simulateurpimsm.domaine.message.*;
import  java.util.Vector;

import  simulateurpimsm.GestionEvenements;

/**
 *  Classe s'occupant de la gestion du protocole PIM SM pour un routeur du domaine
 */
public    class   PimSM
{
   // routeur auquel appartient cet objet PimSm.
   private     Routeur routeur = null ;

   private   TIB tib = null ;

   /**
    * Constructeur:
    */
   public         PimSM (Routeur routeurConteneur)
   {
    routeur = routeurConteneur;
    tib = new TIB();
   }

   /**
    * Traitement d'un message JoinG en provenance de l'interface 'interf'
    */
   public         void  joinG (Interface interf, JoinG joinMsg)
   {
      boolean   needJoin = false  ;

      // Obtention du détail de ce groupe
      GrpTIB detailGroupe = tib.getInfoGroupe(joinMsg.getGroupe());
      if  (detailGroupe == null )
      {
         detailGroupe = tib.newInfoGroupe(joinMsg.getGroupe());
      }

      // Si le flux n'est pas encore traité => besoin d'un Join
      if  ( detailGroupe.getNbrInterface() == 0)
      {
         needJoin = true ;
      }

      // obtention des renseignements propres à l'interface d'entrée pour ce flux
      GrpIntTIB git = detailGroupe.getInterface(interf);
      if  (git == null )
      {
         git = detailGroupe.setInterface(interf);
      }

      // Marquer cette interface comme ayant reçu un Join(*,G)
      git.setJoin(true );

      // Supprimer les éventuels pruneRpt(S,G) arrivés au préalable via cette interface
      int  i = 0;
      int  j = detailGroupe.getNbrInfoSource();
      while  (i < j)
      {
         // obtenir les infos spécifiques à la source traitée
         SrcGrpTIB sgt = detailGroupe.getInfoSource(i);
         if  (sgt != null )
         {
            SrcGrpIntTIB sgit = sgt.getInterface(interf);
            if  (sgit != null )
            {
               // Si cette entrée reste nécessaire
               if  ( sgit.isIgmp() || sgit.isJoin() )
               {
                  sgit.setPruneRPT(false  );
               }
               // Sinon, elle peut-être supprimée
               else
               {
                  sgt.removeInterface(interf);
               }
            }
         }

         i = i + 1;
      }


      // si besoin d'expédier un Join au NextHop routeur
      if  (needJoin)
      {
         // définition du flux (*,G)
         Fluxmulticast flux = new Fluxmulticast();
         flux.setSource(null );
         flux.setGroupe(this .routeur.getDomaine().getGroupeActif(joinMsg.getGroupe()));

         // Emission du Join
         sendJoinPrune(flux, 1);
      }
   }

   /**
    * Traitement d'un message JoinSG en provenance de l'interface 'interf'
    */
   public         void  joinSG (Interface interf, JoinSG joinMsg)
   {
      boolean   needJoin = false  ;

      // Obtention du détail de ce groupe
      GrpTIB detailGroupe = tib.getInfoGroupe(joinMsg.getGroupe());
      if  (detailGroupe == null )
      {
         detailGroupe = tib.newInfoGroupe(joinMsg.getGroupe());
      }

      // Obtention du détail propre à la source pour ce groupe
      SrcGrpTIB sgt = detailGroupe.getInfoSource(joinMsg.getSourceFlux());
      if  ( sgt == null  )
      {
        sgt = detailGroupe.newInfoSource((Station) joinMsg.getSourceFlux());
      }

      // Si le flux n'est pas encore traité => besoin d'un Join
      if  ( sgt.getNbrInterface() == 0)
      {
         needJoin = true ;
      }

      // Obtention du détail de l'interface d'entrée pour ce flux (S,G)
      SrcGrpIntTIB sgit = sgt.getInterface(interf);
      if  (sgit == null )
      {
        sgit = sgt.setInterface(interf);
```

```
    }

    // Marquer cette interface comme ayant reçu un Join(*,G)
    sgit.setJoin(true );


    // si besoin d'expédier un Join au NextHop routeur
    if  (needJoin)
    {
      // définition du flux (S,G)
      Fluxmulticast flux = new Fluxmulticast();
      flux.setSource((Station) joinMsg.getSourceFlux());
      flux.setGroupe(this .routeur.getDomaine().getGroupeActif(joinMsg.getGroupe()));

      // Emission du Join
      sendJoinPrune(flux, 1);
    }
  }

  /**
   * Traitement d'un message PruneG en provenance de l'interface 'interf'
   */
  public        void  pruneG (Interface interf, PruneG pruneMsg)
  {
    // Ce message peut-il être traité
    if  ( isPruneOk(null , pruneMsg.getGroupe(), false  , interf) )
    {
       boolean   needPrune = false  ;

      // Obtention du détail de ce groupe
      GrpTIB detailGroupe = tib.getInfoGroupe(pruneMsg.getGroupe());
      if  (detailGroupe != null )
      {
          // obtention des renseignements propres à l'interface d'entrée pour ce flux
          GrpIntTIB git = detailGroupe.getInterface(interf);
          if  (git != null )
          {
            // Si ce flux est également expédié sur un Lan directement connecté
            if  ( git.isIgmp() )
            {
              // Le flux continue à être expédié via cette interface
              // mais plus à destination d'un autre routeur
              git.setJoin(false  );
            }
            else
            {
              // Le flux ne doit plus être expédié via cette interface
              detailGroupe.removeInterface(interf);
            }
          }

          // Si le flux n'est plus traité
          if  ( detailGroupe.getNbrInterface() == 0)
          {
             // 1) besoin d'un Prune
               needPrune = true ;

             // 2) Suppression des états Prune(S,G,Rpt)
               // Traiter toute les sources spécifiques définies pour ce flux
               int  i = 0;
               int  j = detailGroupe.getNbrInfoSource();
               while  (i < j)
               {
                  // Obtenir la source spécifique à traiter
                  SrcGrpTIB sgt = detailGroupe.getInfoSource(i);
                  if  (sgt != null  )
                  {
                    // Traiter toutes les interfaces définies pour ce flux (S,G)
                    int  i2 = 0;
                    int  j2 = sgt.getNbrInterface();
                    while  (i2 < j2)
                    {
                      SrcGrpIntTIB sgit = sgt.getInterface(i2);
                      if  (sgit != null )
                      {
                        // Si le Flux (S,G) est géré via cette interface
                        if  ( sgit.isIgmp() || sgit.isJoin() )
                        {
                          // Simplement désactivé le PruneRpt
                          sgit.setPruneRPT(false  );
                        }
                        // Si le flux n'est pas géré via cette interface
                        else
                        {
                          // Cette entrée n'est plus nécessaire
                          sgt.removeInterface(sgit.getInterface());
                        }
                      }
                      i2 = i2 + 1;
                    }
                  }
                  i = i + 1;
               }
          }
      }

      // si besoin d'expédier un Prune au NextHop routeur
      if  (needPrune)
      {
        // définition du flux (*,G)
        Fluxmulticast flux = new Fluxmulticast();
        flux.setSource(null );
        flux.setGroupe(this .routeur.getDomaine().getGroupeActif(pruneMsg.getGroupe()));

        // Emission du Prune
        sendJoinPrune(flux, 2);
      }
    }
  }

  /**
   * Traitement d'un message PruneSG en provenance de l'interface 'interf'
   */
  public        void  pruneSG (Interface interf, PruneSG pruneMsg)
  {
    if  ( isPruneOk((Station) pruneMsg.getSourceFlux(), pruneMsg.getGroupe(), false  , interf) )
    {
      boolean   needPrune = false  ;

      // Obtention du détail de ce groupe
      GrpTIB detailGroupe = tib.getInfoGroupe(pruneMsg.getGroupe());
      if  (detailGroupe != null )
      {
        // Obtenir le détail de ce flux (S,G)
        SrcGrpTIB sgt = detailGroupe.getInfoSource(pruneMsg.getSourceFlux());
        if  ( sgt != null )
        {
            // Obtention du détail de l'interface d'entrée pour ce flux (S,G)
            SrcGrpIntTIB sgit = sgt.getInterface(interf);
            if  (sgit != null )
            {
              // Si cette entrée est toujours nécessaire dans la TIB
              if  (sgit.isIgmp() || sgit.isPruneRPT())
              {
                sgit.setJoin(false  );
              }
              else
              {
                // Sinon, cette entrée peut-être supprimée
```

```
                sgt.removeInterface(sgit.getInterface());
              }
            }
        }

       /**
        * Un prune est-il nécessaire ?
        */

        needPrune = true ;

        int  i = 0;
        int  j = detailGroupe.getNbrInfoSource();
        // traiter toutes les sources spécifiques de ce groupe
        while  ( i < j )
        {
            // Obtenir le détail du flux (S,G) traité
            sgt = detailGroupe.getInfoSource(i);
            if  (sgt != null )
            {
                // Obtention du détail de l'interface d'entrée pour ce flux (S,G)
                SrcGrpIntTIB sgit = sgt.getInterface(interf);
                if  (sgit != null )
                {
                  // Si ce flux est toujours demandé via cette interface
                  if  ( sgit.isIgmp() || sgit.isJoin())
                  {
                    needPrune = false  ;
                    i = j;
                  }
                }
            }

            i = i + 1;
        }
      }

      // si besoin d'expédier un PruneSG au NextHop routeur
      if  (needPrune)
      {
        // définition du flux (S,G)
        Fluxmulticast flux = new Fluxmulticast();
        flux.setSource((Station) pruneMsg.getSourceFlux());
        flux.setGroupe(this .routeur.getDomaine().getGroupeActif(pruneMsg.getGroupe()));

        // Emission du Prune
        sendJoinPrune(flux, 2);
      }
    }
}

 /**
  * Traitement d'un message PruneSGRpt en provenance de l'interface 'interf'
  */
 public        void  pruneSGRpt (Interface interf, PruneSGRpt pruneMsg)
 {
    if  ( isPruneOk((Station) pruneMsg.getSourceFlux(), pruneMsg.getGroupe(), true , interf) )
    {
      boolean   needPrune = false  ;

      // Obtention du détail de ce groupe
      GrpTIB detailGroupe = tib.getInfoGroupe(pruneMsg.getGroupe());
      if  (detailGroupe != null )
      {
        // Obtenir le détail de ce flux (S,G)
        SrcGrpTIB sgt = detailGroupe.getInfoSource(pruneMsg.getSourceFlux());
        if  ( sgt == null  )
        {
          sgt = detailGroupe.newInfoSource((Station) pruneMsg.getSourceFlux());
        }

        // Obtention du détail de l'interface d'entrée pour ce flux (S,G)
        SrcGrpIntTIB sgit = sgt.getInterface(interf);
        if  (sgit == null )
        {
           sgit = sgt.setInterface(interf);
        }

        // notification du pruneRpt
        sgit.setPruneRPT(true );

        /**
        * EST-IL NECESSAIRE D'EXPEDIER UN PRUNE RPT VERS LE RP ?
        */

        int  i = 0;
        int  j = detailGroupe.getNbrInterface();
        // S'il y a eu un Join (*,G) expédié au RP
        if  ( i > 0 )
        {
           //Alors, un PruneRpt(S,G) sera peut-être nécessaire
           needPrune = true ;

           // traiter toutes les interface ayant reçu un join (*;G)
           while  ( (i < j) && needPrune )
           {
              // Obtenir l'interface ayant nécessité un Join(*,G) à traiter
              GrpIntTIB git = detailGroupe.getInterface(i);

              // Obtenir le détail pour cette interface propre à la source pour ce groupe
              sgit = sgt.getInterface(git.getInterface());

              // Si ce detail n'existe pas
              if  (sgit == null )
              {
                // Pas besoin de PruneRpt (ce flux est toujours demandé via l'interface traitée)
                needPrune = false  ;
              }
              else
              {
                // Si ce détail n'a pas été créé pour un PruneRpt
                if  ( sgit.isPruneRPT() == false   )
                {
                  // Pas besoin de PruneRpt
                  needPrune = false  ;
                }
              }

              // Traiter l'interfazce suivante
              i = i + i;
           }
        }
      }

      // si besoin d'expédier un PruneRPT au NextHop routeur
      if  (needPrune)
      {
        // définition du flux (S,G)
        Fluxmulticast flux = new Fluxmulticast();
        flux.setSource((Station) pruneMsg.getSourceFlux());
        flux.setGroupe(this .routeur.getDomaine().getGroupeActif(pruneMsg.getGroupe()));

        // Emission du Prune
        sendJoinPrune(flux, 3);
      }
    }
 }

 /**
```

```
  * Traitement d'un message Register en provenance de l'interface 'interf'
  */
 public        void  registerIn (Interface interf, Register registerMsg)
 {
    // destination de ce message
    ElementActif destination = registerMsg.getDestination();

    //Interface menant à la source (!! pas forcément identique à 'interf')
    Interface interfIn = this .routeur.getUnicast().getRoute(registerMsg.getSource());

    // se routeur est-il la destination du message ?
    if  ( this .routeur == destination )
    {
      // Désencapsulation du messageMulticast
      MsgMulticast msgMulticast = (MsgMulticast) registerMsg.getContenu();

      // Obtention de la liste de interface de sortie pour ce message
      Vector lstInterfOut = getListeInterfaceSortie(msgMulticast.getSource(),
                                                    msgMulticast.getGroupe(),
                                                    interfIn);
      // Si cette liste n'est pas null
      if  (lstInterfOut != null )
      {
        // Emission du message multicast sur chaque interface de la liste
        sendMsgMulticast(lstInterfOut, null , msgMulticast);

        // Emission d'un Join (S,G)
        Fluxmulticast flux = new Fluxmulticast();
        flux.setSource((Station) msgMulticast.getSource());
        flux.setGroupe(this .routeur.getDomaine().getGroupeActif(msgMulticast.getGroupe()));
        this .sendJoinPrune(flux, 1);
      }

      // Création d'un Register Stop
      RegisterStop registerStop = new RegisterStop();
      registerStop.setSource(this .routeur);
      registerStop.setDestination(registerMsg.getSource());
      registerStop.setGroupe(registerMsg.getGroupe());
      registerStop.setSourceFlux(registerMsg.getSourceFlux());

      // Notification de l'évenement
      new GestionEvenements().setEvenementMessage(this .routeur, false , interfIn, registerStop);

      // Emission de celui-ci
      interfIn.out(registerStop);
    }
    else
    {
      // Obtention de l'interface menant à la destination
      Interface interfOut = this .routeur.getUnicast().getRoute(destination);

      // Expédition du message
      interfOut.out(registerMsg);
    }
 }

 /**
  * Traitement d'un message RegisterStop en provenance de l'interface 'interf'
  */
 public        void  registerStopIn (Interface interf, RegisterStop registerStopMsg)
 {
    ElementActif destination = registerStopMsg.getDestination();

    // se routeur est-il la destination du message ?
    if  ( this .routeur == destination )
    {
      // Si le flux est en mode Register, Envoyer un Register au RP
      GrpTIB detailGroupe = tib.getInfoGroupe(registerStopMsg.getGroupe());
      if  (detailGroupe != null )
      {
        SrcGrpTIB sgt = detailGroupe.getInfoSource(registerStopMsg.getSourceFlux());
        if  (sgt != null )
        {
          sgt.setRegisterDone(true );
        }
      }
    }
    else
    {
      // Obtention de l'interface menant à la destination
      Interface interfOut = this .routeur.getUnicast().getRoute(destination);

      // Expédition du message
      interfOut.out(registerStopMsg);
    }
 }

 /**
  * Traitement d'un message en transit provenant de l'interface 'interf'
  */
 public        void  transit (Interface interfIn, MsgMulticast msgMulticast)
 {

    // Obtention de la liste des interfaces de sortie pour ce flux
    Vector lstInterfOut = getListeInterfaceSortie(msgMulticast.getSource(),
                                                  msgMulticast.getGroupe(),
                                                  interfIn);

    // Sortie du message via les interfaces de la liste excépté l'interface d'entrée du message
    sendMsgMulticast(lstInterfOut, interfIn, msgMulticast);

    // Si le message provient d'une source directement connectée
    Station stSource = (Station) msgMulticast.getSource();
    if  (interfIn.getConnexion() == stSource.getConnexion())
    {
      // Si ce routeur est le DR de ce Lan
      if  (interfIn.isDr())
      {
        // Si le flux est en mode Register, Envoyer un Register au RP
        GrpTIB detailGroupe = tib.getInfoGroupe(msgMulticast.getGroupe());
        if  (detailGroupe == null )
        {
          sendRegister(msgMulticast);
        }
        else
        {
          SrcGrpTIB sgt = detailGroupe.getInfoSource(msgMulticast.getSource());
          if  (sgt == null )
          {
            sendRegister(msgMulticast);
          }
          else
          {
            if  (sgt.isRegisterDone() == false  )
            {
              sendRegister(msgMulticast);
            }
          }
        }
      }
    }
 }

 /**
  * Sortie d'un message multicast via chaque interface du vecteur lstInterfOut
  * Excepter interfIn
  */
 private        void  sendMsgMulticast (Vector lstInterfOut, Interface interfIn, MsgMulticast msgMulticast)
 {
```

```
    // Traiter toutes les interfaces de la liste
    int  i = 0;
    int  j = 0;

    if  (lstInterfOut != null )
    {
      j = lstInterfOut.size();
    }

    while  ( i < j )
    {
      // obtention  de l'interface
      Interface interfOut = (Interface) lstInterfOut.elementAt(i);

      if  (interfOut != null  )
      {
        // Si l'interface est différente que l'interface d'entrée du message
        if  (interfOut != interfIn )
        {
          // Notification de l'évènement
          new GestionEvenements().setEvenementMessage(this .routeur, false , interfOut, msgMulticast);

          // Expédier le message via cette interface
          interfOut.out(msgMulticast);
        }
      }

      // traiter l'interface suivante
      i = i + 1;
    }
  }

  /**
   * Gestion de l'émission d'un message register en direction du rp du groupe
   * concerné par le message 'msgMulticast'
   */
  private        void  sendRegister (MsgMulticast msgMulticast)
  {
    // Obtention du RP du groupe
    Routeur rp = this .routeur.getDomaine().getRp(msgMulticast.getGroupe());

    // Obtention du détail de ce groupe dans la TIB
    GrpTIB detailGroupe = tib.getInfoGroupe(msgMulticast.getGroupe());
    if  (detailGroupe == null )
    {
      detailGroupe = tib.newInfoGroupe(msgMulticast.getGroupe());
    }

    // Obtention du détail propre à la source
    SrcGrpTIB sgt = detailGroupe.getInfoSource(msgMulticast.getSource());
    if  (sgt == null )
    {
      sgt = detailGroupe.newInfoSource((Station) msgMulticast.getSource());
    }

    // si ce routeur est le RP de ce groupe
    if  (this .routeur == rp)
    {
      // Pas besoin de registering
      sgt.setRegisterDone(true );
    }
    else
    {
      // Création d'un message Register
      Register msgRegister = new Register();
      msgRegister.setSource(this .routeur);
      msgRegister.setDestination(rp);
      msgRegister.setGroupe(msgMulticast.getGroupe());
      msgRegister.setSourceFlux(msgMulticast.getSource());
      msgRegister.setContenu(msgMulticast);

      // Obtention de l'interface menant au RP
      Interface interfOut = this .routeur.getUnicast().getRoute(rp);

      if  (interfOut != null )
      {
        // Notification de l'évenement
        new GestionEvenements().setEvenementMessage(this .routeur, false , interfOut, msgRegister);

        // Expédition du message Register
        interfOut.out(msgRegister);
      }
    }
  }

  /**
   * Gestion d'un message IGMP en provenance d'un Lan pour lequel ce routeur est DR
   *
   * etat = true   Pour une demande de réception
   *      = false  Pour une fin de réception
   *
   * Rem : C'est le Lan qui vérifie que la demande de suppression
   * d'une réception est à prendre en compte.
   */
  public        void  igmp(Fluxmulticast flux, Interface interf, boolean  etat)
  {
    boolean  needJoin = false ;
    boolean  needPrune = false ;

    // Obtention du détail de ce groupe
    GrpTIB detailGroupe = tib.getInfoGroupe(flux.getGroupe().getAdresse());

    // traitement d'une demande de réception
    if  ( etat == true  )
    {
      // si le groupe n'existe pas, le créer
      if  (detailGroupe == null )
      {
        detailGroupe = tib.newInfoGroupe(flux.getGroupe().getAdresse());
      }

      // Si le flux demandé est un flux (*,G)
      if  (flux.getSource() == null )
      {
        // Si le flux n'est pas encore traité => besoin d'un Join
        if  ( detailGroupe.getNbrInterface() == 0)
        {
          needJoin = true ;
        }

        // obtention des renseignements propres à l'interface d'entrée pour ce flux
        GrpIntTIB git = detailGroupe.getInterface(interf);
        if  (git == null )
        {
          git = detailGroupe.setInterface(interf);
        }

        git.setIgmp(true );
      }
      // Si le flux demandé est un flux (S,G)
      else
      {
        // obtention des renseignements propres à cette source pour ce flux
        SrcGrpTIB sgt = detailGroupe.getInfoSource(flux.getSource());
        if  (sgt == null )
        {
          sgt = detailGroupe.newInfoSource(flux.getSource());
        }
```

```
            // Si le flux n'est pas encore traité => besoin d'un Join
            if  ( sgt.getNbrInterface() == 0)
            {
              needJoin = true ;
            }

            // obtention des renseignements propres à l'interface d'entrée pour ce flux
            SrcGrpIntTIB sgit = sgt.getInterface(interf);
            if  ( sgit == null  )
            {
              sgit = sgt.setInterface(interf);
            }
            sgit.setIgmp(true );
        }
    }
    // traitement d'une demande de fin de réception
    else
    {
        // Ne traiter que si le groupe existe
        if  (detailGroupe != null )
        {
          // Si le flux demandé est un flux (*,G)
          if  (flux.getSource() == null )
          {
            // obtention des renseignements propres à l'interface d'entrée pour ce flux
            GrpIntTIB git = detailGroupe.getInterface(interf);
            if  (git != null )
            {
              // Si un le flux est également expédié via cette interface
              // à la suite d'un JOIN
              if  ( git.isJoin() )
              {
                // Simplement désactivé l'expédition pour le LAN
                // (le flux reste actif via cette interface pour le JOIN
                git.setIgmp(false  );
              }
              // Sinon, le flux est expédié via cette interface uniquement
              // pour le LAN à la suite d'un message IGMP-Join
              else
              {
                // Le flux n'est plus actif via cette interface
                detailGroupe.removeInterface(interf);
              }
            }

            // Si le flux n'est plus traité => besoin d'un Prune
            if  ( detailGroupe.getNbrInterface() == 0)
            {
              needPrune = true ;
            }
          }
          // Si le flux demandé est un flux (S,G)
          else
          {
              // obtention des renseignements propres à cette source pour ce flux
              SrcGrpTIB sgt = detailGroupe.getInfoSource(flux.getSource());
              if  (sgt != null )
              {
                // obtention des renseignements propres à l'interface d'entrée pour ce flux
                SrcGrpIntTIB sgit = sgt.getInterface(interf);
                if  ( sgit != null  )
                {
                    // Si un le flux est également expédié via cette interface
                    // à la suite d'un JOIN
                    if  ( sgit.isJoin() )
                    {
                      // Simplement désactivé l'expédition pour le LAN
                      // (le flux reste actif via cette interface pour le JOIN
                      sgit.setIgmp(false  );
                    }
                    // Sinon, le flux est expédié uniquement pour le LAN
                    // à la suite d'un message IGMP
                    else
                    {
                      // Le flux n'est plus actif via cette interface
                      sgt.removeInterface(interf);
                    }
                }

                // Si le flux n'est plus traité => besoin d'un Prune
                if  ( sgt.getNbrInterface() == 0)
                {
                  needPrune = true ;
                }
              }
          }
        }

    if  (needJoin)
    {
      sendJoinPrune(flux, 1);
    }

    if  (needPrune)
    {
      sendJoinPrune(flux, 2);
    }
}

/**
 * Gestion de l'émission d'un Join ou d'un Prune en amont
 * d'un flux (*,G) ou (S,G)
 *
 * Rem : type = 1  => Join
 *       type = 2  => Prune
 *       type = 3  => PruneRpt du flux
 *
 * Rem : Ne vérifie pas le besoin d'expédier ce message.
 *       S'occupe uniquement de la transmition vers le NextHop routeur
 *       du Join OU prune AINSI que des pruneRpt associés à celui-ci
 *       en fonction de l'état de la TIB
 */
private       void  sendJoinPrune(Fluxmulticast flux, int  type)
{
    // Destination finale du message
    ElementDomaine destination = null ;

    // Vérifie s'il s'agit d'un flux (*,g) Ou d'un PruneRpt
    if  ( (flux.getSource() == null ) || (type == 3))
    {
      // la destination est le RP du groupe
      destination = flux.getGroupe().getRp();

      // si la destination est ce routeur
      if  ( destination == this .routeur)
      {
        // le message est à destination
        destination = null ;
      }
    }
    else
    {
      // la destination est la connexion connectée à la source
      destination = flux.getSource().getConnexion();
    }

    // Si le message n'est pas encore arrivé à destination
    if  (destination != null )
```

```
    {
      // recherche de l'interface menant au NextHop routeur
      Interface interf = this .routeur.getUnicast().getRoute(destination);
      if  (interf != null )
      {
         if  (type == 1)
         {
           if  ( flux.getSource() != null  )
           {
/**
 * Traitement d'un Join (S,G)
 * -->
 */
                // N'envoyer un Join(S,G) que si la station source du flux n'est pas directement connectée à ce routeur
                if  (this .routeur.getInterfaceTo((Connexion) destination) == null )
                {
                   // Création du message Join(S,G)
                   JoinSG msg = new JoinSG();
                   msg.setSourceFlux(flux.getSource()); // Source du flux
                   msg.setGroupe(flux.getGroupe().getAdresse()); // Groupe du flux
                   msg.setSource(this .routeur);      // source de ce message
                   msg.setDestination(flux.getSource()); // destination final du message

                   // Notification de l'évenement
                   new GestionEvenements().setEvenementMessage(this .routeur, false , interf, msg);

                   // expédition via l'interface menant au NextHop
                   interf.out(msg);
                 }

                 // Obtention des information concernant le flux (*,G)
                 GrpTIB detailGroupe = tib.getInfoGroupe(flux.getGroupe().getAdresse());
                 if  (detailGroupe != null )
                 {
                  // S'il existe des interface ayant demandé un Join (*,G)
                  if  (detailGroupe.getNbrInterface() > 0 )
                  {
                    // Emission d'un PruneRpt pour le flux (S,G) vers le RP
                    sendJoinPrune(flux, 3);
                  }
                 }
/**
 * <--
 * Fin traitement d'un Join (S,G)
 */
           }
           else
           {
/**
 * Traitement d'un Join (*,G)
 * -->
 */
                 // Création du message Join(*,G)
                 JoinG msg = new JoinG();
                 msg.setGroupe(flux.getGroupe().getAdresse()); // Groupe du flux
                 msg.setSource(this .routeur);      // source de ce message
                 msg.setDestination((ElementActif) destination); // destination final du message

                 // Notification de l'évenement
                 new GestionEvenements().setEvenementMessage(this .routeur, false , interf, msg);

                 // expédition via l'interface menant au NextHop
                 interf.out(msg);

                 // Obtention des informations concernant le flux (*,G)
                 GrpTIB detailGroupe = tib.getInfoGroupe(flux.getGroupe().getAdresse());
                 if  (detailGroupe != null )
                 {
                    // Parcourir toute les Sources définies pour ce groupe
                    int  i = 0;
                    int  j = detailGroupe.getNbrInfoSource();
                    while  (i < j)
                    {
                        // Obtenir la source spécifique à traiter
                        SrcGrpTIB sgt = detailGroupe.getInfoSource(i);
                        if  (sgt != null  )
                        {
                          boolean  needPruneRpt = false  ;
                          int  i2 = 0;
                          int  j2 = sgt.getNbrInterface();
                          while  ( i2 < j2 )
                          {
                                // Obtention de l'interface à traiter
                                SrcGrpIntTIB sgit = sgt.getInterface(i2);
                                if  ( sgit != null )
                                {
                                  // Si un Join (S,G) à été traité pour ce flux
                                  if  ( sgit.isIgmp() || sgit.isJoin() )
                                  {
                                    // Ce flux est traité via le SPT
                                    needPruneRpt = true ;

                                    // il n'est pas nécessaire de vérifier les interfaces suivantes
                                    i2 = j2;
                                  }
                                }

                                // Interface suivante
                                i2 = i2 + 1;
                          }

                          // Si ce Flux (S,G) est traité via le SPT
                          if  ( needPruneRpt )
                          {
                              // Définition du flux (S,G)
                              Fluxmulticast fluxRpt = new Fluxmulticast();
                              fluxRpt.setSource(sgt.getSource());
                              fluxRpt.setGroupe(flux.getGroupe());

                              // Expédition d'un pruneRpt (S,G)
                              sendJoinPrune(fluxRpt, 3);
                          }
                        }

                       // Source spécifique suivante
                       i = i + 1;
                    }
                 }
/**
 * <--
 * Fin traitement d'un Join (*,G)
 */
           }
         }
         else  // type != 1
         {  if  (type == 2)
            {
                if  ( flux.getSource() != null  )
                {
/**
 * Traitement d'un Prune (S,G)
 * -->
 */
                  // N'envoyer un Prune(S,G) que si la station source du flux n'est pas directement connectée à ce routeur
                  if  (this .routeur.getInterfaceTo((Connexion) destination) == null )
                  {
                     // Création du message Prune(S,G)
                     PruneSG msg = new PruneSG();
                     msg.setSourceFlux(flux.getSource()); // Source du flux
```

```
                    msg.setGroupe(flux.getGroupe().getAdresse()); // Groupe du flux
                    msg.setSource(this .routeur);      // source de ce message
                    msg.setDestination(flux.getSource()); // destination final du message

                    // Notification de l'évenement
                    new GestionEvenements().setEvenementMessage(this .routeur, false , interf, msg);

                    // expédition via l'interface menant au NextHop
                    interf.out(msg);
                  }

                  // Obtention des informations concernant le flux (*,G)
                  GrpTIB detailGroupe = tib.getInfoGroupe(flux.getGroupe().getAdresse());
                  if (detailGroupe != null )
                  {
                   // S'il existe des interfaces ayant demandé un Join (*,G)
                   // (==> un PruneRpt (S,G) à été émis qu'il faut supprimer )
                   if (detailGroupe.getNbrInterface() > 0 )
                   {
                     // Définition d'un flux (*,G)
                     Fluxmulticast fluxRp = new Fluxmulticast();
                     fluxRp.setGroupe(flux.getGroupe());

                     // Emission d'un Join pour le flux (S,G) vers le RP
                     //(Celui-ci sera suivi des PruneRpt des sources encore actives)
                     sendJoinPrune(fluxRp, 1);
                   }
                  }
/**
 * <--
 * Fin traitement d'un Prune (S,G)
 */
              }
              else
              {
/**
 * Traitement d'un Prune (*,G)
 * -->
 */
                  // Création du message Prune(*,G)
                  PruneG msg = new PruneG();
                  msg.setGroupe(flux.getGroupe().getAdresse()); // Groupe du flux
                  msg.setSource(this .routeur);      // source de ce message
                  msg.setDestination((ElementActif) destination); // destination final du message

                  // Notification de l'évenement
                  new GestionEvenements().setEvenementMessage(this .routeur, false , interf, msg);

                  // expédition via l'interface menant au NextHop
                  interf.out(msg);
/**
 * <--
 * Fin traitement d'un Prune (*,G)
 */
              }
          }
          else  // type != 2
          {
 /**
 * Traitement d'un PruneRpt (S,G)
 * -->
 */
                  // Création du message PruneRpt(S,G)
                  PruneSGRpt msg = new PruneSGRpt();
                  msg.setSourceFlux(flux.getSource()); // Source du flux
                  msg.setGroupe(flux.getGroupe().getAdresse()); // Groupe du flux
                  msg.setSource(this .routeur);      // source de ce message
                  msg.setDestination((ElementActif) destination); // destination final du message

                  // Notification de l'évenement
                  new GestionEvenements().setEvenementMessage(this .routeur, false , interf, msg);

                  // expédition via l'interface menant au NextHop
                  interf.out(msg);
      /**
       * <--
       * Fin traitement d'un PruneRpt (S,G)
       */
            } // fin type != 2
          } // fin type != 1
        } // fin interf != null
      } // fin destination != null
  }


  /**
   * Vérifie si CE routeur peut élaguer le flux (source, groupe)
   * de l'interface interf
   *
   * - Pour un prune(*,G)       : définir source = null
   * - Pour un pruneRpt (S,G)   : définir (source, groupe) et mettre rpt à true
   *
   *
   * Rem : Cette méthode est utilisée par les différentes méthode Prune
   * et ce, afin de simuler la temporisation de la prise en compte d'un prune
   * en provenance d'un Lan
   */
  private        boolean   isPruneOk(Station source, AdresseIp groupe, boolean   rpt, Interface interf)
  {
    // si l'interface est connectée à un LAN, il faut vérifier
    if  ( interf.getConnexion() instanceof     Lan )
    {
      // Création du flux
      Fluxmulticast flux = new Fluxmulticast();
      flux.setSource(source);
      flux.setGroupe(routeur.getDomaine().getGroupeActif(groupe));

      // demande de vérification au lan
      Lan lan = (Lan) interf.getConnexion();
      return   lan.VerificationPruneOk(flux, rpt, interf);
    }

    // Si l'interface n'est pas connectée à un Lan, le prune peut être exécuté.
    return   true ;
  }

  /**
   * Vérifie si le flux (source, groupe) est attendu en entrée
   * par l'interface interf
   *
   * Rem : Cette méthode est utilisée par ce routeur
   * suite à la demande d'un Lan
   * (Celle-ci faisant elle même suite à la demande de la méthode isPruneOk
   * d'un routeur de ce Lan
   */
  public         boolean   isFluxNecessaire(Fluxmulticast flux, boolean   rpt, Interface interf)
  {
    // Obtention du détail du groupe
    GrpTIB detailGroupe = tib.getInfoGroupe(flux.getGroupe().getAdresse());
    if  (detailGroupe != null )
    {
      if  ( flux.getSource() == null )            // ==> Prune(*,G)
      {
        // Si interf est l'interface menant au RP du groupe
        Interface interfRp = this .routeur.getUnicast().getRoute(flux.getGroupe().getRp());
        if  (interf == interfRp)
        {
          // Si un Join(*,G) est toujours actif sur au moins une interface
          if  (detailGroupe.getNbrInterface() < 0 )
```

```
            {
              return  true ;
            }
          }
        }
        else
        {
          // obtenir les infos propres à cette source  pour ce groupe
          SrcGrpTIB sgt = detailGroupe.getInfoSource(flux.getSource());
          if  (sgt != null  )
          {
            if  (rpt == false  )                    // ==> Prune(S,G)
            {
              // Si interf est l'interface menant à la source
              Interface interfSrc = this .routeur.getUnicast().getRoute(flux.getSource());
              if  (interf == interfSrc)
              {
                // Si ce flux(*,G) est toujours actif sur cette interface
                int  i = 0;
                int  j = sgt.getNbrInterface();
                while  (i < j)
                {
                  // Obtention de l'interface
                  SrcGrpIntTIB sgit = sgt.getInterface(i);

                  if  (sgit != null )
                  {
                    // Si ce flux(S,G) est toujours actif sur cette interface
                    if  (sgit.isIgmp() || sgit.isJoin())
                    {
                      return  true ;
                    }
                  }

                  i = i + 1;
                }
              }
            }
            else                                    // ==> Prune(S,G,rpt)
            {
              // Si interf est l'interface menant au RP du groupe
              Interface interfRp = this .routeur.getUnicast().getRoute(flux.getGroupe().getRp());
              if  (interf == interfRp)
              {
                // traiter toutes les interfaces ayant reçu un Join(*,G)
                int  i = 0;
                int  j = detailGroupe.getNbrInterface();
                while  (i < j)
                {
                  // Obtenir l'interface traitée
                  GrpIntTIB git = detailGroupe.getInterface(i);

                  if  (git != null )
                  {
                    // Obtenir du détail de cette interface propre à la Source S pour ce groupe
                    SrcGrpIntTIB sgit = sgt.getInterface(git.getInterface());

                    // si ce flux(S,G,rpt) est toujours nécessaire via cette interface
                    if  ( sgit == null )
                    {
                      return  true ;
                    }
                    else
                    {
                      if  ( sgit.isPruneRPT() == false  )
                      {
                        return  true ;
                      }
                    }
                  }
                }
              }
            }
          }

          return  false ;
        }

        /**
         * Obtention de la liste des interface de sortie d'un flux (S,G)
         * entrant dans ce routeur via l'interface 'interfEntree'
         * (Pour un flux (*,G) définir S = null)
         *
         * Retourne null si cette liste est vide (C'est-à-dire si la TIB est vide
         * pour ce flux OU si l'interface d'entrée n'est pas la bonne pour ce flux)
         *
         */
        private        Vector getListeInterfaceSortie(ElementActif source, AdresseIp groupe, Interface interfEntree)
        {
          // Création de la liste d'interface
          Vector listeInterface = null ;

          // Obtention du détail de ce groupe
          GrpTIB detailGroupe = tib.getInfoGroupe(groupe);

          // Si ce groupe existe dans la TIB
          if  ( detailGroupe != null )
          {
            // Obtention de l'interface menant à la source
            Interface interf = routeur.getUnicast().getRoute(source);

            // Si l'interface d'entrée mène à la source
            if  (interf == interfEntree)
            {
              // Obtention de la liste des interfaces via le SP-Tree
              listeInterface = OIListSG(source, detailGroupe);
            }

            // Si l'interface d'entrée ne mène pas à la source
            if  ( listeInterface == null )
            {
              // Obtention du RP du groupe
              Routeur rp = routeur.getDomaine().getRp(groupe);

              if  (rp != null )
              {
                // Obtention de l'interface menant au RP du groupe
                interf = routeur.getUnicast().getRoute(rp);

                // Si l'interface d'entrée mène au RP
                if  (interf == interfEntree)
                {
                    // Obtention de la liste des interfaces via le RP-Tree
                    listeInterface = OIListG(source, detailGroupe);
                }
              }
            }
          }

          // retourne la liste d'interface crée.
          return  listeInterface;
        }

        /**
         * Creation de la liste des interfaces de sortie pour le flux (*,groupe)
         * en provenance de la source 'source'
         *
```

```
   * Retourne la liste NON VIDE des interfaces OU null
   *
   *  !! Ne vérifie pas le mode de gestion du flux !!
   */
  private   Vector OIListG(ElementActif source, GrpTIB groupe)
  {
     Vector liste = new Vector();

     // détail propre à cette source pour ce Groupe
     SrcGrpTIB sgt = groupe.getInfoSource(source);

     int  i = 0;
     int  j = groupe.getNbrInterface();

     //traiter toutes les interfaces
     while  (i < j)
     {
       // obtention de l'interface traitée
       GrpIntTIB git = (GrpIntTIB) groupe.getInterface(i);
       if  (git != null )
       {
         // Si un IGMP Qwery est actif
         if  ( git.isIgmp() )
         {
           // Ajout de l'interface à la liste
           liste.addElement(git.getInterface());
         }
         else
         {
           // Si un Join(*,g) est actif
           if  ( git.isJoin() )
           {
             // Et qu'il n'y a pas eu de Prune(S,G,rpt), Ajout de l'interface à la liste
             SrcGrpIntTIB sgit = null ;
             if  (sgt != null )
             {
               sgit = sgt.getInterface(git.getInterface());
             }
             if  ( sgit == null  )
             {
               liste.addElement(git.getInterface());
             }
             else
             {
               if  (sgit.isPruneRPT() == false  )
               {
                 liste.addElement(git.getInterface());
               }
             }
           }
         }
       }

       // passer à l'interface suivante
       i = i + 1;
     }

     if (liste.isEmpty())
     {
       return  null ;
     }
     return  liste;
  }

  /**
   * Création de la liste des interfaces de sortie pour le flux (source,groupe)
   * en provenance de la source 'source'
   *
   * Retourne la liste NON VIDE des interfaces OU null
   *
   * !! Ne vérifie pas le mode de gestion du flux !!
   */
  private   Vector OIListSG(ElementActif source, GrpTIB groupe)
  {
     // Obtention de la liste des interface de sorties pour le flux(*,G)
     Vector liste = OIListG(source, groupe);
     if  (liste == null  )
     {
       liste = new Vector();
     }

     // Ajouter à cette liste les interfaces n'étant pas encore reprises
     // et ayant demandé la réception du flux (S,G)
     SrcGrpTIB sgt = groupe.getInfoSource(source);

     int  i = 0;
     int  j = 0;
     if  (sgt != null )
     {
       j = sgt.getNbrInterface();
     }

     //traiter toutes les interfaces
     while  (i < j)
     {
       // obtention de l'interface traitée
       SrcGrpIntTIB sgit = (SrcGrpIntTIB) sgt.getInterface(i);
       if  (sgit != null )
       {
         // Si le flux est demandé sur cette interface
         if  (sgit.isIgmp() || sgit.isJoin())
         {
           Interface interf = sgit.getInterface();

           if  (liste.contains(interf) != true )
           {
             liste.addElement(interf);
           }
         }
       }

       // passer à l'interface suivante
       i = i + 1;
     }

     if (liste.isEmpty())
     {
       return  null ;
     }
     return  liste;
  }


/**                                                                          **
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                         */


  /**
   * Affichage de la table de routage multicast
   */
  public   String toString ()
  {
     String res = new String();

     res = res + "- TIB PimSm  \n";

     res = res + this .tib;
```

```
        return res;
    }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  simulateurpimsm.domaine.message.*;
import  java.util.Vector;
import  java.io.*;

import  simulateurpimsm.GestionEvenements;

/**
 * Classe représentant un routeur du domaine
 */
public    class   Routeur extends   ElementActif implements   Serializable
{
   // Nombre maximum d'interfaces intégrées dans ce type d'élément
   protected      int  maxInterface = 10;

   // ce routeur est-il Candidat BSR (Non par défaut) ?
   protected      boolean  isCBSR = false  ;

   // ce routeur est-il Candidat RP (Non par défaut) ?
   protected      boolean  isCRP = false  ;

   // Objet permettant l'implémentation du protocol PIM SM
   protected      PimSM pimSm = null  ;

   // Gestion du routage unicast
   protected      Unicast unicast = null  ;

   // Variables statiques permettant l'attribution automatique d'un num aux routeurs
   protected      static    int  nextNumRouteur = 0;

   // Num attribué à ce routeur lors de sa création
   protected      int  numRouteur = 0;


   /**
   * Constructeur:
   */
   public   Routeur()
   {
    maxInterfaces = 10;

    // initialisation du numéro du routeur
    nextNumRouteur = nextNumRouteur + 1;
    numRouteur = nextNumRouteur;

    // initialisation du routage unicast
    unicast = new Unicast();

    // initialisation du routage multicast
    initPimSm();
   }

   /**
   * Initilisation PimSm
   */
   public      void  initPimSm()
   {
     pimSm = new PimSM(this  );
   }
```

```
   /**
    * Réception du message 'newMessage' en entrée dans le routeur
    * via l'interface 'interf'
    */
   public         void  in (Message newMessage, Interface interf)
   {
      // Notification de l'événement
      new GestionEvenements().setEvenementMessage(this  , true  , interf, newMessage);

      // Le message est-il du type Join ?
      if  (newMessage.getType().equals("Join") )
      {
          // Est-ce un Join (*,G) ?
          if  (newMessage.getSousType().equals("G") )
          {
            // transférer le message au protocole Pim SM
            JoinG joinG = (JoinG) newMessage;
            pimSm.joinG(interf, joinG);

            //Fin de traitement
            return  ;
          }

          // sinon, est-ce un Join (S,G) ?
          if  (newMessage.getSousType().equals("SG") )
          {
            // transférer le message au protocole Pim SM
            JoinSG joinSG = (JoinSG) newMessage;
            pimSm.joinSG(interf, joinSG);
          }

          //Fin de traitement
          return  ;
      }

      // Le message est-il du type Prune ?
      if  (newMessage.getType().equals("Prune") )
      {
          // Est-ce un Prune (*,G) ?
          if  (newMessage.getSousType().equals("G") )
          {
            // transférer le message au protocole Pim SM
            PruneG pruneG = (PruneG) newMessage;
            pimSm.pruneG(interf, pruneG);

            //Fin de traitement
            return  ;
          }

          // sinon, est-ce un Prune (S,G) ?
          if  (newMessage.getSousType().equals("SG") )
          {
            // transferer le message au protocole Pim SM
            PruneSG pruneSG = (PruneSG) newMessage;
            pimSm.pruneSG(interf, pruneSG);

            //Fin de traitement
            return  ;
          }

          // sinon, est-ce un Prune (S,G,Rpt) ?
          if  (newMessage.getSousType().equals("SGRpt") )
          {
            // transférer le message au protocole Pim SM
            PruneSGRpt pruneSGRpt = (PruneSGRpt) newMessage;
            pimSm.pruneSGRpt(interf, pruneSGRpt);
          }

          //Fin de traitement
```

```
            return ;
        }

        // Le message est-il du type Multicast NON traité ci-dessus ?
        // C'est à dire un simple message multicast en transit
        if (newMessage instanceof MsgMulticast)
        {
            // transférer le message au protocole Pim SM
            MsgMulticast msgMulti = (MsgMulticast) newMessage;
            pimSm.transit(interf, msgMulti);

            //Fin de traitement
            return ;
        }

    // Fin de traitement des messages Multicast.
    // A ce stade, c'est un message unicast.

        // Ce message unicast est-il un message Pim SM Register ?
        if (newMessage instanceof Register )
        {
            // transférer le message au protocole Pim SM
            Register register = (Register) newMessage;
            pimSm.registerIn(interf, register);

            //Fin de traitement
            return ;
        }

        // Sinon, ce message unicast est-il un message Pim SM RegisterSop ?
        if (newMessage instanceof RegisterStop )
        {
            // transférer le message au protocole Pim SM
            RegisterStop registerStop = (RegisterStop) newMessage;
            pimSm.registerStopIn(interf, registerStop);

            //Fin de traitement
            return ;
        }

        // Est-ce un message unicast en transit ?
        if (newMessage.getDestination() != this )
        {
            // Obtention de l'interface menant à cette destination
            Interface intDest = unicast.getRoute(newMessage.getDestination());

            // Si cette Interface existe, sortir le message
            if ( intDest != null )
            {
              out(newMessage, interf);
            }
            //Fin de traitement
            return ;
        }

        // SInon, C'est un message unicast a destination de ce routeur ?
        // AUCUN TRAITEMENT ACTUELLEMENT REALISE POUR CE GENRE DE MESSAGE
    }

    /**
     * Gestion d'une demande IGMP
     *
     * Rem : Est utilisée par un Lan pour lequel ce routeur est DR
     */
    public void igmp (Fluxmulticast flux, Interface interf, boolean etat)
    {
        // Notification de l'évenement
        new GestionEvenements().setEvenementIgmp((Lan)interf.getConnexion(), this , etat, flux.getSource(), flux.getGroupe(
).getAdresse());
        pimSm.igmp(flux, interf, etat);
    }

    /**
     * Configure ce routeur pour être Candidat BSR ou non
     */
    public void setCBSR (boolean etat)
    {
     isCBSR= etat;
    }

    /**
     * Ce routeur est-il Candidat BSR ?
     */
    public boolean isCBSR ()
    {
     return isCBSR;
    }

    /**
     * Configure ce routeur pour être Candidat RP ou non
     */
    public void setCRP (boolean etat)
    {
     isCRP = etat;
    }

    /**
     * Ce routeur est-il Candidat RP ?
     */
    public boolean isCRP ()
    {
     return isCRP;
    }

    /**
     * retourne la table de routage unicast de ce routeur.
     */
    public Unicast getUnicast ()
    {
     return unicast;
    }

    /**
     * Interroge le protocole PIM SM pour savoir si le flux 'flux' via
     * l'interface d'entrée 'interf' est toujours actif sur ce routeur.
     */
    public boolean isFluxNecessaire(Fluxmulticast flux, boolean rpt, Interface interf)
    {
       return pimSm.isFluxNecessaire(flux, rpt, interf);
    }

    /**
     * Retourne la plus petite adresse Ip de toutes les interfaces de ce routeur
     */
    public AdresseIp getLowerIp()
    {
       AdresseIp adr = null ;

       // Parcourir toutes les interfaces de ce routeur
       int i = 0;
       int j = this .getNbrInterfaces();
       while (i < j)
       {
         // Obtenir l'interface à traiter
         Interface interf = this .getInterface(i);

         if (interf != null )
         {
```

```
        AdresseIp adrInterf = interf.getAdresse();
        if  (adrInterf != null )
        {
          // si aucune adresse déjà sélectionnée
          if  (adr == null )
          {
            // Sélectionner celle-ci
            adr = adrInterf;
          }
          else
          {
            // Si celle-ci est plus petite que l'adresse sélectionnée
            if  ( adrInterf.isLower(adr) )
            {
              // Sélectionner celle-ci
              adr = adrInterf;
            }
          }
        }

        // passer à l'interface suivante
        i = i + 1;
      }

      return  adr;
    }



/**                                                                          **
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                         */


    /**
     * Affichage du nom de ce routeur
     */
    public  String toString ()
    {
       return  new String("Rt." + new Integer(this .numRouteur));
    }

    /**
     * Configuration détaillée de ce routeur
     */
    public  String  toStringConfig ()
    {
       String res = this .toString() + " [ CBSR=" + new Boolean(this .isCBSR) + " / CRP=" + new Boolean(this .isCRP) + " ]\n";

       int  i = 0;
       int  j = this .getNbrInterfaces();
       if  (j > 0 )
       {
         res = res + "- Interface(s) :\n";
         while  (i < j)
         {
           Interface interf = this .getInterface(i);
           String via = new String ("");
           if  (interf != null )
           {
             Connexion connect = interf.getConnexion();
             if  ( connect instanceof     PaP)
             {
               PaP connectPaP = (PaP) connect;
               via = "" + connectPaP.linkedTo(interf) + " Via ";
             }

             res = res + "    " + interf.toStringConfig() + " Connectée à " + via + connect + "\n";
           }
           i = i + 1;
         }
       }

       res = res + this .unicast;
       res = res + this .pimSm;

       return  res;
    }


/**                                                                          **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                          */

    /**
     * Sauvegarde de cette instance de l'objet
     */
    private  void  writeObject(ObjectOutputStream s) throws  IOException
    {
      // Sauvegarde des attributs de cette instance de routeur
      s.writeInt(this .maxInterface);
      s.writeInt(this .nextNumRouteur);
      s.writeInt(this .numRouteur);
      s.writeBoolean(this .isCBSR);
      s.writeBoolean(this .isCRP);

      // Sauvegarde des données propres aux éléments actifs
      s.writeInt(this .nextNumInterf);

      // Sauvegarde des données propres aux Elements du domaine
      s.writeInt(this .maxInterfaces);
      s.writeObject(this .interfaceContenue);
      s.writeObject(this .domaine);
    }

    /**
     * Restauration de l'instance de l'objet
     */
    private  void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
    {
      // Recupération des attributs de cette instance de routeur
      this .maxInterface = s.readInt();
      this .nextNumRouteur = s.readInt();
      this .numRouteur = s.readInt();
      this .isCBSR = s.readBoolean();
      this .isCRP = s.readBoolean();

      // Recupération des données propres aux éléments actifs
      this .nextNumInterf = s.readInt();

      // Recupération des données propres aux Elements du domaine
      this .maxInterfaces = s.readInt();
      this .interfaceContenue = (Vector) s.readObject();
      this .domaine = (Domaine) s.readObject();

      // Création d'une nouvelle table de routage unicast
      this .unicast = new Unicast();
    }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;

/**
 * Objet contenant toutes les informations propres à une interface pour un
 * flux multicast (S,G) spécifique de la TIb d'un routeur Pim SM
 */
public    class   SrcGrpIntTIB
{
   private      boolean   igmp = false  ;
   private      boolean   join = false  ;

   /**
    * Interface référencée par cette instance
    */
   private    Interface interf = null  ;

   /**
    * Un message Prune(S,G,rpt) est-il arrivé pour ce flux (S,G) ?
    * via l'interface de cette instance
    */
   private      boolean   pruneRPT = false  ;

   /**
    * Constructeur
    */
   public   SrcGrpIntTIB(Interface newInterface)
   {
      interf = newInterface;
   }

   /**
    * Obtention de l'interface désignée par cette entrée
    */
   public   Interface getInterface()
   {
      return   interf;
   }

   /**
    * Définit l'arrivée d'un Message IGMP Join par cette interface en provenance
    * d'un LAN pour lequel ce routeur est DR
    */
   public        void  setIgmp (boolean   etat)
   {
      igmp = etat;
   }

   /**
    * Un message IGMP join est-il arrivé par cette interface en provenance d'un
    * LAN pour lequel ce routeur est DR ?
    */
   public        boolean   isIgmp ()
   {
      return   igmp;
   }

   /**
    * Défini l'arrivée d'un Join par cette interface en provenance d'un routeur
    */
   public        void  setJoin (boolean   etat)
   {
      join = etat;
   }

   /**
    * Un Join est-il arrivé en provenance d'un autre routeur par cette interface ?
    */
   public        boolean   isJoin ()
   {
      return   join;
   }

   /**
    * Définit si un PruneRPT est arrivé pour ce flux
    */
   public        void  setPruneRPT (boolean   etat)
   {
      pruneRPT = etat;
   }

   /**
    * Un PruneRPT est-il arrivé pour ce flux ?
    */
   public        boolean   isPruneRPT ()
   {
      return   pruneRPT;
   }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;

/**
 * Objet contenant toutes les informations de la TIb d'un routeur Pim SM
 * concernant une flux multicast (S,G) spécifique
 */
public    class   SrcGrpTIB
{
   /**
    * Source du flux
    * (Le groupe d'épend de l'instance de l'objet GrpTib
    * contenant cette instance)
    */
   private      Station source = null ;

   /**
    * Ce flux (S,G) est-il géré via le SPT par le DR ?
    */
   private      boolean   modeSPT = false  ;

   /**
    * Gestion du mode Register
    * d'un routeur (DR) pour ce flux
    */
   private      boolean   registerDone = false  ;

   private   Vector lstInterface  = new Vector();

   /**
    * Définition de la source référencée par cette ligne de la TIB
    */
   public         void  setSource (Station newSource)
   {
      source = newSource;
   }

   /**
    * Obtention de la source référencée par cette ligne de la TIB
    */
   public         Station getSource ()
   {
      return   source;
   }

   /**
    * Définition du mode Register pour ce flux
    * (Utilisé par un DR)
    */
   public         void    setRegisterDone(boolean   etat)
   {
      registerDone = etat;
   }

   /**
    * Ce flux est-il en mode Register On?
    * (utilisé par un DR)
    */
   public         boolean   isRegisterDone()
   {
      return   registerDone;
   }

   /**
    * Définit si ce flux est en mode SPT
    */
   public         void  setModeSpt (boolean   etat)
   {
      modeSPT = etat;
   }

   /**
    * Ce flux est-il en mode SPT ?
    */
   public         boolean   isModeSpt ()
   {
      return   modeSPT;
   }

   /**
    * Définit si un PruneRPT est arrivé pour ce flux
    */
   public         void  setPruneRPT (boolean   etat, Interface interf)
   {
      // Obtention du détail de l'interface
      SrcGrpIntTIB sgit = getInterface(interf);

      // si le détail est obtenu, configurer le pruneRPT
      if  ( sgit != null )
      {
        sgit.setPruneRPT(etat);
      }
   }

   /**
    * Un PruneRPT est-il arrivé pour ce flux via l'interface 'interf'?
    */
   public         boolean   isPruneRPT (Interface interf)
   {
      // Obtention du détail de l'interface
      SrcGrpIntTIB sgit = getInterface(interf);

      // si le détail est obtenu, retourner son pruneRPT
      if  ( sgit != null )
      {
        return   sgit.isPruneRPT();
      }

      // Le détail de l'iterface n'existe pas
      return   false  ;
   }

   /**
    * Ajoute une interface à la liste des interfaces de ce flux
    */
   public         SrcGrpIntTIB setInterface (Interface newInterface)
   {
      // Recherche de cette interface dans la liste des interfaces
      int   i = 0;
      int   j = lstInterface.size();

      // traité toute la liste
      while   (i < j)
      {
        // Obtenir l'interface traitée
        SrcGrpIntTIB sgit = (SrcGrpIntTIB) lstInterface.elementAt(i);

        // si l'interface traitée est la même que newInterface ne pas continuer
        if  ( sgit.getInterface() == newInterface )
```

```
      {
        return  sgit;
      }

      // Traiter l'interface suivante
      i = i + 1;
    }

    // l'interface n'existe pas => l'ajouter
    SrcGrpIntTIB sgit = new SrcGrpIntTIB(newInterface);
    lstInterface.addElement(sgit);
    return  sgit;
  }

  /**
   * suppression d'une interface de la liste des interfaces par lesquelles
   * le flux de ce groupe doit-être transféré
   */
  public        void  removeInterface (Interface interf)
  {
    // Recherche de cette interface dans la liste des interfaces
    int  i = 0;
    int  j = lstInterface.size();

    // traité toute la liste
    while  (i < j)
    {
      // Obtenir l'interface traitée
      SrcGrpIntTIB sgit = (SrcGrpIntTIB) lstInterface.elementAt(i);

      // si l'interface traitée est la même que 'interf'
      if  ( sgit.getInterface() == interf )
      {
        lstInterface.removeElementAt(i);
        return ;
      }

      // Traiter l'interface suivante
      i = i + 1;
    }

  }

  /**
   * Obtention du nombre d'interface de ce flux
   */
  public        int  getNbrInterface ()
  {
    return  lstInterface.size();
  }

  /**
   * Obtention du détail pour ce flux de l'interface d'indice 'indice'
   * ( Retourne null si l'indice est trop grand )
   */
  public        SrcGrpIntTIB getInterface (int  indice)
  {
    // si l'indice est trop grand, retourne null
    if  ( indice > getNbrInterface())
    {
      return  null ;
    }

    return  (SrcGrpIntTIB) lstInterface.elementAt(indice);
  }

  /**
   * Obtention du détail pour ce flux de l'interface 'interf'
   * ( Retourne null si l'interface n'est pas trouvée )
   */
  public        SrcGrpIntTIB getInterface (Interface interf)
  {
    // traiter toutes les interfaces
    int  i = 0;
    int  j = getNbrInterface();
    while  (i < j)
    {
      // Obtention du détail interface traitée
      SrcGrpIntTIB sgit = (SrcGrpIntTIB) lstInterface.elementAt(i);

      // si l'objet est celui recherché, le retourner
      if  (sgit.getInterface() == interf)
      {
        return  sgit;
      }

      // traiter l'interface suivante
      i = i + 1;
    }

    // l'interface n'a pas été trouvée
    return  null ;
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  simulateurpimsm.domaine.message.*;
import  java.util.Vector;
import  java.io.*;

/**
 * Classe représentant une station de travail du domaine
 */
public    class   Station extends   ElementActif implements   Serializable
{
   // Constantes :
   public   static   final   boolean   EMIS = true ;
   public   static   final   boolean   RECUS = false  ;

   // Variables statiques permettant une attribution automatique à cette station
   protected      static   int  nextNumStation = 0;

   protected      Vector fluxRecus = null ;
   protected      Vector fluxEmis = null ;

   protected       int  numStation = 0;

   /**
   * Constructeur:
   */
   public   Station()
   {
    nextNumStation = nextNumStation + 1;
    numStation = nextNumStation;

    // une station ne possède qu'une interface
    maxInterfaces = 1;

    // préparation des vecteurs flux
    fluxRecus = new Vector();
    fluxEmis = new Vector();
   }

   /**
    * Réception d'un message en provenance de l'interface 'interf'
    */
   public        void  in (Message newMessage, Interface interf)
   {
   }

   /**
    * Ajout d'un flux (*,G) à réceptionner par cette station
    */
   public        void  addRecever (GroupeMulticast groupe)
   {
      // Recherche du flux
      Fluxmulticast fm = getFlux(RECUS, groupe);

      // Si le flux n'existe pas
      if  ( fm == null )
      {
        // le créer
        fm = new Fluxmulticast();
        fm.setGroupe(groupe);
        // l'ajouter à la liste
        fluxRecus.addElement(fm);

        // Lancer un IGMP Qwery pour ce flux
        sendIgmp(fm, true );
      }
   }


   /**
    * Ajout d'un flux (S,G) à réceptionner par cette station
    */
   public        void  addRecever (Station source, GroupeMulticast groupe)
   {
      // Recherche du flux
      Fluxmulticast fm = getFlux(RECUS, groupe, source);

      // Si le flux n'existe pas
      if  ( fm == null )
      {
        // le créer
        fm = new Fluxmulticast();
        fm.setGroupe(groupe);
        fm.setSource(source);

        // l'ajouter à la liste
        fluxRecus.addElement(fm);

        // Lancer un IGMP Qwery pour ce flux
        sendIgmp(fm, true );
      }
   }

   /**
    * Suppression de la réception d'un flux (*,G) multicat par cette station
    */
   public        void  removeRecever (GroupeMulticast groupe)
   {
      // Recherche du flux
      Fluxmulticast fm = getFlux(RECUS, groupe);

      // Si le flux existe
      if  ( fm != null )
      {
        // le supprimer de la liste
        fluxRecus.removeElement(fm);

        // Lancer un IGMP Prune pour ce flux
        sendIgmp(fm, false  );
      }
   }

   /**
    * Suppression de la réception d'un flux multicat par cette station
    */
   public        void  removeRecever (Station source, GroupeMulticast groupe)
   {
      // Recherche du flux
      Fluxmulticast fm = getFlux(RECUS, groupe, source);

      // Si le flux existe
      if  ( fm != null )
      {
        // le supprimer de la liste
        fluxRecus.removeElement(fm);

        // Lancer un IGMP Prune pour ce flux
        sendIgmp(fm, false  );
      }
```

```
  }

  /**
   * Initialisation du protocol PimSm
   */
  public       void  initPimSm()
  {
     // Expédition d'un IGMP Qwery pour chaque flux réceptionné par cette station
     int  i = 0;
     int  j = fluxRecus.size();
     while  (i < j)
     {
       // obtenir le flux traité
       Fluxmulticast flux = (Fluxmulticast) fluxRecus.elementAt(i);

       if  (flux != null )
       {
         sendIgmp(flux, true );
       }

       // traiter le flux suivant
       i = i + 1;
     }

     // Expédition d'un premier message pour chaque flux emis par cette station
     i = 0;
     j = fluxEmis.size();
     while  (i < j)
     {
       // obtenir le flux traité
       Fluxmulticast flux = (Fluxmulticast) fluxEmis.elementAt(i);

       if  (flux != null )
       {
         this .sendMsgMulticastIgmp(flux.getGroupe());
       }

       // traiter le flux suivant
       i = i + 1;
     }
  }

  /**
   * gestion de l'émission de message IGMP pour un flux donné
   *
   * Rem : etat = true     pour une demande de réception
   *       etat = false    pour une demande de fin de réception
   */
  public       void  sendIgmp(Fluxmulticast flux, boolean   etat)
  {
    // obtention de l'interface de sortie
    Interface interf = this .getInterface(0);
    if  ( interf != null )
    {
      Connexion connexion = interf.getConnexion();
      if  (connexion != null )
      {
        if  (connexion instanceof     Lan)
        {
          Lan lan = (Lan) connexion;
          lan.igmp(flux, etat, interf);
        }
      }
    }
  }

  /**
   * gestion de l'émission d'un message multicast pour un flux donné
   */
    public       void  sendMsgMulticastIgmp(GroupeMulticast groupe)
    {
      // obtention de l'interface de sortie
      Interface interf = this .getInterface(0);
      if  ( interf != null )
      {
        // le flux est-il géré par cette station ?
        Fluxmulticast fm = this .getFlux(EMIS, groupe);
        if  (fm != null )
        {
          // Création du message
          MsgMulticast msg = new MsgMulticast();
          msg.setSource(this );
          msg.setGroupe(groupe.getAdresse());
          msg.setContenu(new String ("Message Multicast de " + this .toString() + " en direction du groupe " +
groupe.getAdresse()));

          // Emission via l'interface de la station
          interf.out(msg);
        }
      }
    }

    /**
     * Ajout d'un flux vers lequel cette station doit émettre
     */
    public       void  addSource (GroupeMulticast groupe)
    {
      // Recherche du flux
      Fluxmulticast fm = getFlux(EMIS, groupe);

      // Si le flux n'existe pas
      if  ( fm == null )
      {
        // le créer
        fm = new Fluxmulticast();
        fm.setGroupe(groupe);

        // l'ajouter à la liste
        fluxEmis.addElement(fm);
      }

      // émission d'un premier paquet vers ce groupe
      this .sendMsgMulticastIgmp(groupe);
    }

    public       void  removeSource (GroupeMulticast groupe)
    {
      // Recherche du flux
      Fluxmulticast fm = getFlux(EMIS, groupe);

      // Si le flux existe
      if  ( fm != null )
      {
        // le supprimer de la liste
        fluxEmis.removeElement(fm);
      }
    }

   /**
    * Recherche d'un flux (*,G) géré par cette station
    * (Si le flux n'existe pas, return null)
    */
    public   Fluxmulticast getFlux(boolean   type, GroupeMulticast groupe)
    {
      return   getFlux (type, groupe, null );
    }

   /**
    * Recherche d'un flux géré par cette station
```

```
  * (Si le flux n'existe pas, return null)
  */
  public   Fluxmulticast getFlux(boolean   type, GroupeMulticast groupe, Station source)
  {
     // Sélection du vecteur de recherche
     Vector flux = null ;
     if  (type == EMIS)
     {
       flux = fluxEmis;
     }
     else
     {
       flux = fluxRecus;
     }

     int  i = 0;
     int  j = flux.size();
     while  ( i < j )
     {
       // sélectionner le flux
       Fluxmulticast fm = (Fluxmulticast) flux.elementAt(i);

       // Le flux est-il celui demandé ?
       if  ( fm.getGroupe() == groupe && fm.getSource() == source )
       {
         return  fm;
       }
       // traiter le flux suivant
       i = i + 1;
     }

     // le flux n'est pas trouvé
     return  null ;
  }

  /**
   * Obtention de la connexion à laquelle cette station est connectée
   * si la station n'est pas connectée, return null
   */
  public   Connexion getConnexion()
  {
     Interface interf = this .getInterface(0);
     if  ( interf != null )
     {
       return  interf.getConnexion();
     }
     return  null ;
  }

 /**                                                                    **
  * --------  Méthodes utilisées pour l'interfacage utilisateur ---------- *
  **                                                                     */

 public   String toString ()
 {
   return  new String ( "St." + numStation );
 }

  /**
   * Configuration détaillée de cette Station
   */
  public   String  toStringConfig ()
  {
     String res = this .toString();

     String connect = new String ("NON DEFINI");
     if  (this .getConnexion() != null )
     {
       connect = ""+ this .getConnexion();
     }

     String interf = new String ("NON DEFINI");
     if  ( this .getInterface(0) != null )
     {
       interf = this .getInterface(0).toStringConfig();
     }

     res = res + " [ Interface : " + interf + " / Connexion : " + connect + " ]\n";

     int  i = 0;
     int  j = this .fluxEmis.size();
     if  (j > 0 )
     {
       res = res + "- Flux Multicast Emis :\n";
       while  (i < j)
       {
         Fluxmulticast flux = (Fluxmulticast) this .fluxEmis.elementAt(i);
         if  (flux != null )
         {
           res = res + "    " + flux.getGroupe() + "\n";
         }
         i = i + 1;
       }
     }

     i = 0;
     j = this .fluxRecus.size();
     if  (j > 0 )
     {
       res = res + "- Flux Multicast Réceptionné :\n";
       while  (i < j)
       {
         Fluxmulticast flux = (Fluxmulticast) this .fluxRecus.elementAt(i);
         if  (flux != null )
         {
           res = res + "    " + flux + "\n";
         }
         i = i + 1;
       }
     }
     return  res;
   }


/**                                                                     **
 * --------  Méthodes utilisées pour la sérialisation ---------------------- *
 *                                                                      */

  /**
   * Sauvegarde de cette instance de l'objet
   */
  private   void  writeObject(ObjectOutputStream s) throws  IOException
  {
    // Sauvegarde des attributs de cette instance de station
    s.writeInt(this .nextNumStation);
    s.writeInt(this .numStation);
    s.writeObject(this .fluxEmis);
    s.writeObject(this .fluxRecus);

    // Sauvegarde des données propres aux éléments actifs
    s.writeInt(this .nextNumInterf);

    // Sauvegarde des données propres aux éléments du domaine
    s.writeInt(this .maxInterfaces);
    s.writeObject(this .interfaceContenue);
    s.writeObject(this .domaine);
  }

  /**
```

```
   * Restauration de l'instance de l'objet
   */
  private   void  readObject(ObjectInputStream s) throws  IOException, ClassNotFoundException
  {
    // Sauvegarde des attributs de cette instance de station
    this .nextNumStation = s.readInt();
    this .numStation = s.readInt();
    this .fluxEmis = (Vector) s.readObject();
    this .fluxRecus = (Vector) s.readObject();

    // Recupération des données propres aux éléments actifs
    this .nextNumInterf = s.readInt();

    // Recupération des données propres aux Elements du domaine
    this .maxInterfaces = s.readInt();
    this .interfaceContenue = (Vector) s.readObject();
    this .domaine = (Domaine) s.readObject();
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package simulateurpimsm.domaine;

import java.util.Vector;

/**
 * Objet représentant la TIB d'un routeur Pim SM
 * ( Cet objet ainsi que les autres objets inclus dans cette TIB ne font
 * office que de conteneurs. Il appartient aux méthodes de l'objet PimSM
 * de gérer ces informations.)
 */
public class TIB
{
   /**
    * Vecteur contenant chaque ligne de la TIB spécifique à un groupe
    * (Instances de l'objet GrpTIB)
    */
   private Vector lstGroupe = null ;

   /**
    * Constructeur:
    */
   public TIB()
   {
      init();
   }

   /**
    * Initialisation la TIB
    */
   public void init()
   {
      lstGroupe = new Vector();
   }

   /**
    * Obtention des informations de routage concernant un groupe multicast.
    * (si le groupe n'existe pas dans cette TIB, retourne null)
    */
   public GrpTIB getInfoGroupe (AdresseIp adrGroupe)
   {
      int i = 0;
      int j = lstGroupe.size();

      // traiter tous les groupes de cette TIB
      while (i < j)
      {
        // obtenir le groupe traité
        GrpTIB gt = (GrpTIB) lstGroupe.elementAt(i);

        // Si l'adresse du groupe traité est celle demandée, retourner le groupe
        if ( gt.getAdrGroupe() == adrGroupe )
        {
          return gt;
        }

        // Passer au groupe suivant
        i = i + 1;
      }

      // le groupe demandé n'a pas été trouvé.
      return null ;
   }

   /**
    * Ajout d'un nouveau groupe à cette TIB et le retourne
    * (si le groupe existe déja dans cette TIB, il n'est pas ajouté )
    */
   public GrpTIB newInfoGroupe (AdresseIp newAdresseIp)
   {
      // Tentative de retrouver le groupe dans cette TIB
      GrpTIB gp = getInfoGroupe(newAdresseIp);

      // si le groupe n'existe pas, le créer
      if ( gp == null )
      {
        // Création du groupe
        gp = new GrpTIB();
        gp.setAdrGroupe(newAdresseIp);

        // Ajout à cette TIB
        lstGroupe.addElement(gp);
      }

      // retourner le groupe
      return gp;
   }

   /**
    * Suppression d'un groupe dans cette TIB
    */
   public void removeInfoGroupe (AdresseIp newAdresseIp)
   {
      // Tentative de retrouver le groupe dans cette TIB
      GrpTIB gp = getInfoGroupe(newAdresseIp);

      // si le groupe est trouvé, le supprimer
      if ( gp != null )
      {
        lstGroupe.removeElement(gp);
      }
   }


/**                                                                          **
 * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                          */


   /**
    * Affichage de cette TIB
    */
   public String toString ()
   {
      String res = new String("");
      int i = 0;
      int j = lstGroupe.size();
      if ( j == 0 )
      {
        res = new String("-> (Table vide)\n");
      }
      while (i < j)
      {
        // obtenir le groupe traité
        GrpTIB gt = (GrpTIB) lstGroupe.elementAt(i);

        if ( gt != null )
        {
          res = res + "    " + new Integer (i) + ") flux( * , " + gt.getAdrGroupe() + ")\n";

          // Afficher toutes les interfaces
```

```
          int  iInterf = 0;
          int  jInterf = gt.getNbrInterface();
          while  (iInterf < jInterf )
          {
            GrpIntTIB git = gt.getInterface(iInterf);
            if  ( git != null )
            {
              res = res + "     -> " + git.getInterface() + " [ Join=" + new Boolean(git.isJoin()) + " / Igmp=" + new
Boolean(git.isIgmp()) + " ]\n";
            }
            iInterf = iInterf + 1 ;
          }

          // Afficher toutes les sources spécifiques
          int  iSrc = 0;
          int  jSrc = gt.getNbrInfoSource();
          while  (iSrc < jSrc )
          {
            SrcGrpTIB sgt = gt.getInfoSource(iSrc);
            if  ( sgt != null )
            {
              res = res + "    " + new Integer (i) + "." + new Integer (iSrc) + ") flux( "+ sgt.getSource() + " , " +
gt.getAdrGroupe() + ") [ RegDone=" + new Boolean(sgt.isRegisterDone()) + "]\n";

              // Afficher toutes les interfaces de cette source spécifique
              int  iScrInt = 0;
              int  jScrInt =sgt.getNbrInterface();
              while  (iScrInt < jScrInt)
              {
                SrcGrpIntTIB sgit = sgt.getInterface(iScrInt);
                if  (sgit != null )
                {
                  res = res + "     -> " + sgit.getInterface() + " [ Join=" + new Boolean(sgit.isJoin()) + " / Igmp=" + new
Boolean(sgit.isIgmp()) + " / PruneRpt=" + new Boolean(sgit.isPruneRPT()) + " ]\n";
                }
                iScrInt = iScrInt + 1;
              }
            }
            iSrc = iSrc + 1 ;
          }

        }

        i = i + 1;
      }
      return  res;
    }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

/**
 *  Classe référencant une interface contenue dans une ligne d'une matrice de routage.
 *  Ainsi que le poids de cette liaison.
 *
 *  Celle-ci utilisée via la matrice de routage
 */
public    class  ToInterface
{
   // Interface référencée par cette instance
   private   Interface interf = null ;

   // Poids de la liaison
   private   int  poids;

   /**
    * Constructeur
    */
   public     ToInterface()
   {
    poids = 0;
   }

   /**
    * Configuration de l'interface concernée par cette instance
    */
   public        void  setInterface (Interface newInterface)
   {
      interf = newInterface;
   }

   /**
    * Obtention de l'interface concernée par cette instance
    */
   public        Interface getInterface ()
   {
      return   interf;
   }

   /**
    * Configuration du poids de cette connexion
    */
   public         void  setPoids (int  newPoids)
   {
      poids = newPoids;
   }

   /**
    * Obtention du poids de cette liaison
    */
   public         int  getPoids ()
   {
      return   poids;
   }
}
```

8.4.3. les messages

(Package simulateurpimsm.domaine.message)

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package  simulateurpimsm.domaine.message;

/**
 * Classe représentant un message Join émis en direction d'un groupe multicast
 * par un élément actif du domaine
 */
public  class  JoinG extends  MsgMulticast
{
  /**
   * Constructeur:
   */
  public  JoinG()
  {
    type =new String("Join");
    sousType = new String("G");
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm.domaine.message;

/**
 * Classe représentant un message Join émis en direction d'un flux multicast
 * émis par une source spécifique par un élément actif du domaine.
 */
public class JoinSG extends JP_SG
{
  /**
   * Constructeur:
   */
  public JoinSG()
  {
    type =new String("Join");
    sousType = new String("SG");
  }
}
```

```java
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package  simulateurpimsm.domaine.message;

/**
 * Classe représentant un message Join émis en direction d'un flux multicast
 * émis par une source spécifique par un DR du domaine lors d'un changement de mode.
 */
public  class  JoinSGRpt extends  JP_SG
{
  /**
   * Constructeur:
   */
  public  JoinSGRpt()
  {
    type =new String("Join");
    sousType = new String("SGRpt");
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm.domaine.message;

import simulateurpimsm.domaine.ElementActif;
import simulateurpimsm.domaine.PimSmException;

/**
 * Abstraction représentant un message Join ou Prune émis en amont
 * d'un Flux multicast spécifique par un élément actif du domaine
 */
abstract public class JP_SG extends MsgMulticast
{
  // élément actif ayant émis le message
  ElementActif sourceFlux = null ;

  /**
   * Constructeur:
   */
  public JP_SG()
  {
  }

  /**
   * Méthode :       setSourceFlux
   * Paramètre :     ElementActif adresse
   * Retour :        /
   * Description :   Permet de définir la source du message
   */
  public void setSourceFlux(ElementActif srcFlux)
  {
    sourceFlux = srcFlux;
  }

  /**
   * Méthode:      getSourceFlux
   * Retour :      sourceFlux ou null si la source du flux multicast n'est pas définie
   * Description: retourne la source du flux multicast du message
   */
  public ElementActif getSourceFlux()
  {
    return sourceFlux;
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Eric Dupuis
 * @version      1.0
 */

package simulateurpimsm.domaine.message;

import simulateurpimsm.domaine.ElementActif;

/**
 * Classe représentant un message IP émis par un élément actif du domaine à destination
 * d'un autre élément actif du domaine
 */
public class Message
{
  // élément actif ayant émis le message
  ElementActif source = null ;

  // élément actif auquel le message est destiné
  ElementActif destination = null ;

  // définition du type et du sous type du message
  String type = null ;
  String sousType = null ;

  // Contenu du message
  Object contenu = null ;

  /**
   * Constructeur:
   */
  public Message()
  {
    type = new String("Unicast");
    sousType = new String("Standard");
  }

  /**
   * Méthode :      setSource
   * Paramètre :    ElementActif adresse
   * Retour :       /
   * Description :  Permet de définir la source du message
   */
  public void setSource(ElementActif src)
  {
    source = src;
  }

  /**
   * Méthode:     getSource
   * Retour :     source ou null si la source n'est pas définie
   * Description: retourne la source du message
   */
  public ElementActif getSource()
  {
    return source;
  }

  /**
   * Méthode :      setDestination
   * Paramètre :    ElementActif adresse
   * Retour :       /
   * Description :  Permet de définir la destination du message
   */
  public void setDestination(ElementActif dest)
  {
    destination = dest;
  }

  /**
   * Méthode:     getDestination
   * Retour :     destination ou null si la destination n'est pas définie
   * Description: retourne la destination du message
   */
  public ElementActif getDestination()
  {
    return destination;
  }

  /**
   * Méthode:     getType
   * Retour :     type
   * Description: retourne le type du message
   */
  public String getType()
  {
    return type;
  }

  /**
   * Méthode:     getSousType
   * Retour :     sousType
   * Description: retourne le sous type du message
   */
  public String getSousType()
  {
    return sousType;
  }

  /**
   * Méthode :      setContenu
   * Paramètre :    Objet
   * Retour :       /
   * Description :  Permet de définir la source du message
   */
  public void setContenu(Object cnt)
  {
    contenu = cnt;
  }

  /**
   * Méthode:     getContenu
   * Retour :     Contenu ou null si le contenu n'est pas défini
   * Description: retourne la contenu du message
   */
  public Object getContenu()
  {
    return contenu;
  }

  /**                                                                    **
   * --------  Méthodes utilisées pour l'interfacage utilisateur ------------ *
   **                                                                     */

  public String toString ()
  {
    return new String ( "Message IP : " +
                        "\n  - Source : " + source +
                        "\n  - Destination : " + destination +
                        "\n  - Type : " + type +
                        "\n  - Sous Type : " + sousType );
  }


}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */
package  simulateurpimsm.domaine.message;

import  simulateurpimsm.domaine.AdresseIp;
import  simulateurpimsm.domaine.PimSmException;

/**
 * Classe représentant un message Multicast émis par un élément actif du domaine à destination
 * d'un groupe multicast
 */
public  class  MsgMulticast extends  Message
{
  // Adresse IP du groupe multicast concerné par le message Register
  AdresseIp groupe = null ;

  /**
   * Constructeur:
   */
  public  MsgMulticast()
  {
    super ();
    type = new String("Multicast");
  }

  /**
   * Méthode :       setGroupe
   * Paramètre :     Adresse Ip multicast du groupe
   * Retour :        /
   * Description :   Permet de définir l'adresse Ip du groupe concerné par ce message
   */
  public  void  setGroupe(AdresseIp grp)
  {
    groupe = grp;
  }

  /**
   * Méthode:      getGroupe
   * Retour :      Groupe ou null s'il n'est pas définie
   * Description: retourne le groupe multicast concerné par ce message
   */
  public  AdresseIp getGroupe()
  {
    return  groupe;
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package  simulateurpimsm.domaine.message;

/**
 * Classe représentant un messagePrune émis en direction d'un groupe multicast
 * par un élément actif du domaine à destination
 */
public  class  PruneG extends  MsgMulticast
{
  /**
   * Constructeur:
   */
  public  PruneG()
  {
    type =new String("Prune");
    sousType = new String("G");
  }
}
```

```java
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package simulateurpimsm.domaine.message;

/**
 * Classe représentant un message Prune émis en direction d'un flux multicast
 * émis par une source spécifique par un élément actif du domaine.
 */
public class PruneSG extends JP_SG
{
  /**
   * Constructeur:
   */
  public PruneSG()
  {
    type =new String("Prune");
    sousType = new String("SG");
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */

package  simulateurpimsm.domaine.message;

/**
 * Classe représentant un message Prune émis par un DR du domaine lors
 * d'un changement de mode en direction du RP d'un groupe
 * et concernant une source spécifique d'un flux multicast
 */
public  class  PruneSGRpt extends  JP_SG
{
  /**
   * Constructeur:
   */
  public  PruneSGRpt()
  {
    type =new String("Prune");
    sousType = new String("SGRpt");
  }
}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Eric Dupuis
 * @version      1.0
 */
package  simulateurpimsm.domaine.message;

import  simulateurpimsm.domaine.AdresseIp;
import  simulateurpimsm.domaine.*;

 /**
 * Classe représentant un message Register émis par un élément actif du domaine à destination
 * du RP du groupe multicast concerné
 */
public   class   Register extends   Message
{
  // Adresse IP du groupe multicast concerné par le message Register
  AdresseIp groupe = null ;

  // élément actif ayant émis le message
  ElementActif sourceFlux = null ;

  /**
   * Constructeur:
   */
  public   Register()
  {
    super ();
    sousType = new String("Register");
  }

  /**
   * Méthode :       setGroupe
   * Paramètre :     Adresse Ip multicast du groupe
   * Retour :        /
   * Description :   Permet de définir l'adrese Ip du groupe concerné par ce message
   */
  public   void  setGroupe(AdresseIp grp)
  {
    groupe = grp;
  }

  /**
   * Méthode:      getGroupe
   * Retour :      Groupe ou null s'il n'est pas définie
   * Description: retourne le groupe multicast concerné par ce message
   */
  public    AdresseIp getGroupe()
  {
    return   groupe;
  }

  /**
   * Méthode :       setSourceFlux
   * Paramètre :     ElementActif adresse
   * Retour :        /
   * Description :   Permet de définir la source du message
   */
  public   void  setSourceFlux(ElementActif srcFlux)
  {
    sourceFlux = srcFlux;
  }

  /**
   * Méthode:      getSourceFlux
   * Retour :      sourceFlux ou null si la source du flux multicast n'est pas définie
   * Description: retourne la source du flux multicast du message
   */
  public    ElementActif getSourceFlux()
  {
    return   sourceFlux;
  }

  /**
   * Méthode :       setContenu
   * Paramètre :     Message
   * Retour :        /
   * Description :   Permet d'encapsuler un message multicast dans ce message Register
   */
  public   void  setContenu(Message cnt)
  {
    super .setContenu( (Object) cnt);
  }

  /**
   * Méthode:      getContenu
   * Retour :      Contenu ou null si le contenu n'est pas défini
   * Description: retourne la contenu du message
   */
  public    Message getMsgContenu()
  {
    return   (Message) contenu;
  }

}
```

```
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author
 * @version 1.0
 */
package  simulateurpimsm.domaine.message;

import  simulateurpimsm.domaine.message.*;
import  simulateurpimsm.domaine.*;

/**
 * Classe représentant un message Register Stop émis par le RP du groupe
 * multicast concerné du domaine à destination du élément actif
 */
public  class  RegisterStop extends  Message
{
  // Adresse IP du groupe multicast concerné par le message Register
  AdresseIp groupe = null ;

  // élément actif ayant émis le message
  ElementActif sourceFlux = null ;

  /**
   * Constructeur:
   */
  public  RegisterStop()
  {
    super ();
    sousType = new String("RegisterStop");
  }

  /**
   * Méthode :      setGroupe
   * Paramètre :    Adresse Ip multicast du groupe
   * Retour :       /
   * Description :  Permet de définir l'adrese Ip du groupe concerné par ce message
   */
  public  void  setGroupe(AdresseIp grp)
  {
    groupe = grp;
  }

  /**
   * Méthode:     getGroupe
   * Retour :     Groupe ou null s'il n'est pas défini
   * Description: retourne le groupe multicast concerné par ce message
   */
  public  AdresseIp getGroupe()
  {
    return  groupe;
  }

  /**
   * Méthode :      setSourceFlux
   * Paramètre :    ElementActif adresse
   * Retour :       /
   * Description :  Permet de définir la source du message
   */
  public  void  setSourceFlux(ElementActif srcFlux)
  {
    sourceFlux = srcFlux;
  }

  /**
   * Méthode:     getSourceFlux
   * Retour :     sourceFlux ou null si la source du flux multicast n'est pas définie
   * Description: retourne la source du flux multicast du message
   */
  public  ElementActif getSourceFlux()
  {
    return  sourceFlux;
  }

}
```

```java
/**
 * Title:        Simulateur PIM SM
 * Description:  Simulateur du protocol PIM SM (Protocol Independant Multicast Sparse Mode)
 * Copyright:    Copyright (c) 2000
 * Company:
 * @author       Dupuis Eric
 * @version      1.0
 */

package  simulateurpimsm.domaine;

import  java.util.Vector;

/**
 * Classe implémentant la gestion du protocole de routage unicast
 * (Elle est ici relativement simple dans la mesure
 *  ou le calcul des routes est effectué par l'objet domaine)
 */
public    class   Unicast
{
   // Vecteur contenant toutes les Lignes de routage de cette table de routage unicast
   protected    Vector tableRoutage  = new Vector();


   /**
   * Constructeur:
   */
   public   Unicast()
   {
    init();
   }

  /**
   * initialise l'objet pour une nouvelle utilisation
   */
   public         void  init ()
   {
    tableRoutage = new Vector();
   }

  /**
   * Méthode:      getLigneRoutage
   * Retour :      LigneRoutage pointant sur l'élément demandé ou null si cette ligne n'existe pas dans cette table
   * Description:  recherche d'une ligne pointant sur un élément du domaine dans la table de routage
   */
   private       LigneRoutage getLigneRoutage(ElementDomaine element)
   {
    LigneRoutage ligne = null ;
    int  i = 0;

    // traiter toutes les lignes de la table
    int  to = tableRoutage.size();
    while  (i < to)
    {
      // obtention de la ligne traitée
      ligne = (LigneRoutage) tableRoutage.get(i);

      // si l'élément pointé dans la ligne traitée est celui demandé, retourner cette ligne
      if  (ligne.getElement() == element )
      {
        return  ligne;
      }

      //passer à l'élément suivant
      i = i + 1;
    }

   // aucune ligne de routage ne pointe sur l'élément recherché.
   return  null ;
  }

 /**
  * Méthode:      setRoute
  * Retour :      /
  * Description:  Ajout d'une nouvelle ligne de routage à la table
  *               ou remplacement de la ligne pointant sur newElement si celle-ci est déja existante
  */
  public        void  setRoute (ElementDomaine newElement, Interface newInterface)
  {
   // recherche d'une ligne de routage dans la table pointant déja sur cet élément
   LigneRoutage ligne = getLigneRoutage (newElement);

   // Si cette ligne de routage existe déja, la supprimer
   if  ( ligne != null  )
   {
     tableRoutage.removeElement(ligne);
   }

   // Création de la ligne de routage
   ligne = new LigneRoutage(newElement, newInterface);

   // Ajout de cette nouvelle ligne de routage à la table
   tableRoutage.addElement(ligne);
  }

 /**
  * Méthode:      getRoute
  * Retour :      Interface associée à elementDomaine s'il existe une ligne dans la table de routage pointant sur celui-ci
  *               sinon null
  * Description:  retourne l'interface de sortie menant à un élément du domaine
  */
  public        Interface getRoute (ElementDomaine elementDomaine)
  {
   // Si l'élément est une station
   if  (elementDomaine instanceof    Station)
   {
     Station st = (Station) elementDomaine;
     elementDomaine = st.getConnexion();
   }
   // recherche d'une ligne de routage dans la table pointant déja sur cet élément
   LigneRoutage ligne = getLigneRoutage (elementDomaine);

   // Si la ligne de routage existe
   if  ( ligne != null  )
   {
     return  ligne.getInterface();
   }

   // aucune ligne n'a été trouvée
   return  null ;
  }


/**                                                                    **
 * -------- Méthodes utilisées pour l'interfacage utilisateur ------------ *
 **                                                                    */


  /**
   * Affichage de la table de routage unicast
   */
  public  String toString ()
  {
     String res = new String();
     int  i = 0;
     int  j = this .tableRoutage.size();

     res = res + "- Table de routage Unicast : Nbr de d'entrée = " + new Integer(j) + "\n";
```

```
      while  (i < j)
      {
        // obtention de la ligne traitée
        LigneRoutage ligne = (LigneRoutage) tableRoutage.get(i);
        if  (ligne != null )
        {
          res = res + "    " + new Integer(i) + ") " + ligne.getElement() + " via " + ligne.getInterface() + "\n";
        }
        i = i + 1;
      }

      return  res;
    }
}
```